Applications of MDE

Jean Bézivin

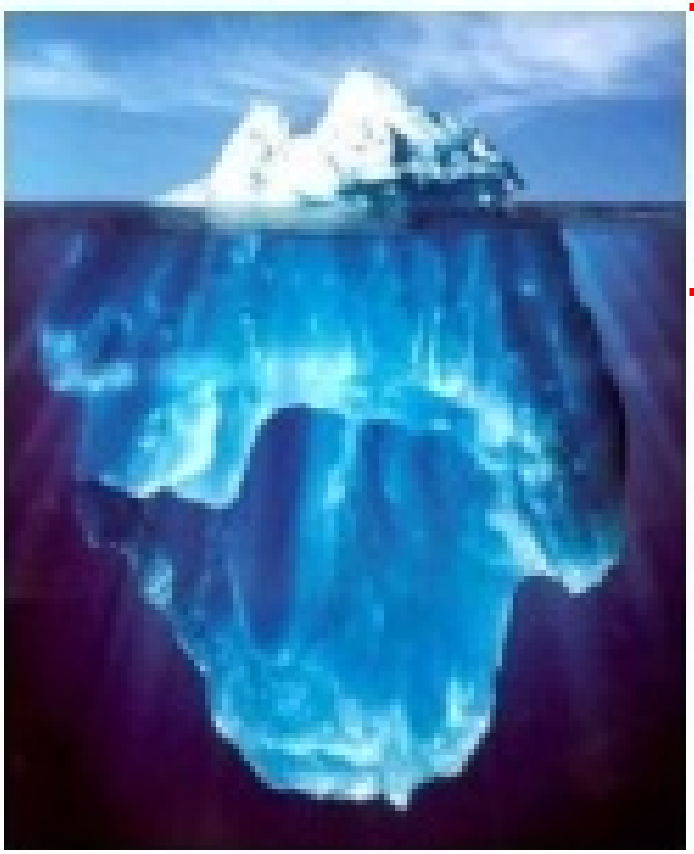
JBezivin@gmail.com

Lecture 3: Applications of MDE

- Classification
  - of MDE approaches
    - software (code)
      - software production
      - software maintenance
      - software operation
    - complex systems
    - etc.
  - of models and metamodels
  - of systems
  - of transformations
  - of operations on models
  - of tools
  - etc.

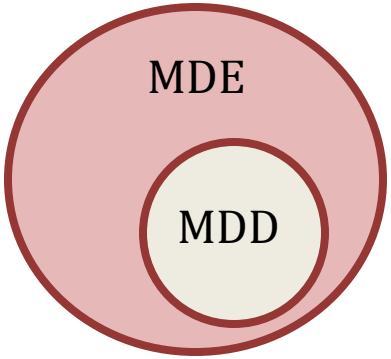
Broadening application area

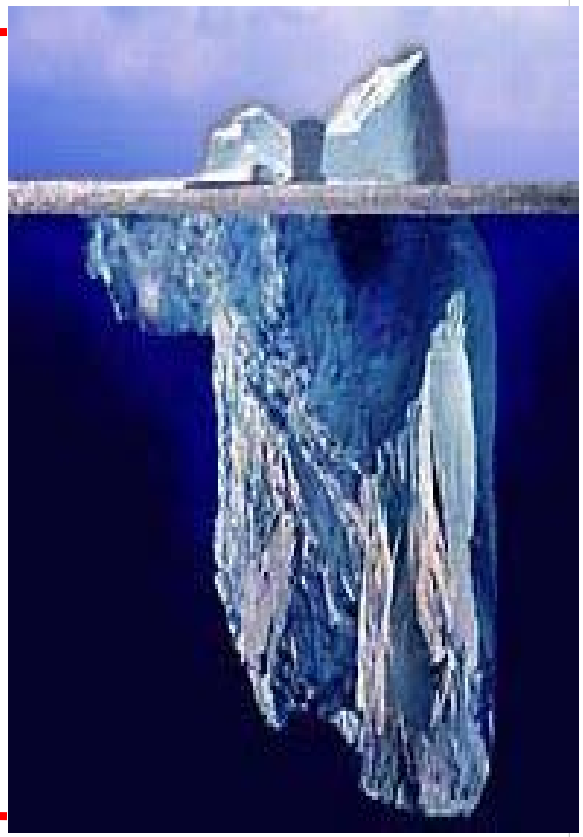
We have not yet seen the full application deployment of MDE



MDD

MDE

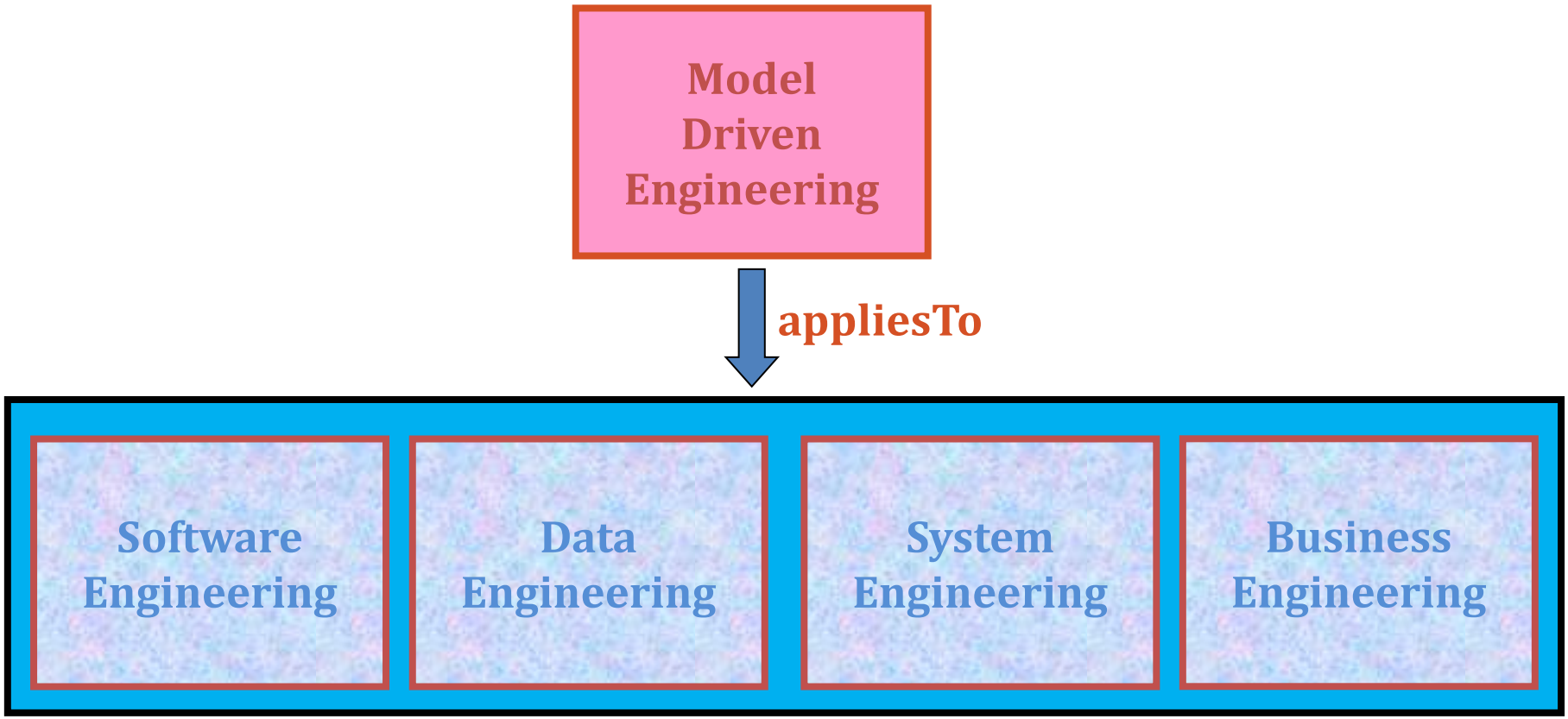




MDD = Model Driven Development

MDE = Model Driven Engineering

MDE applies to major IT fields

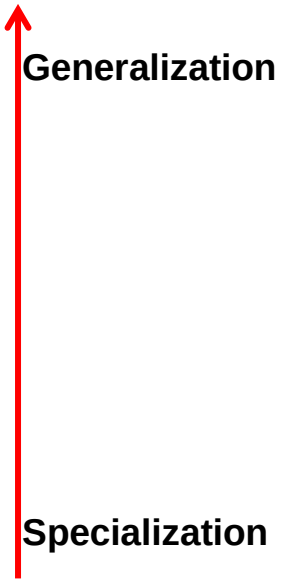


Applicability scope of model driven engineering

- Model Driven Engineering (MDE) is frequently presented as an important change in software development practices.
- Behind this new trend, one may recognize a lot of different objectives and solutions.
- The simple translation from a UML platform-independent model to a Java-encoded platform specific application represents only a very limited and naïve initial use case.
- This lecture studies the multiple facets of MDE and its evolution in the recent period.
- Among the various extended possibilities, one may mention model driven reverse engineering, stream-based and data-centric model based system organization, or various usages of models at run-time.
- Also everyday new areas of application of MDE are being identified.
- In order to understand this, we need to survey the various classification of MDE artifacts.

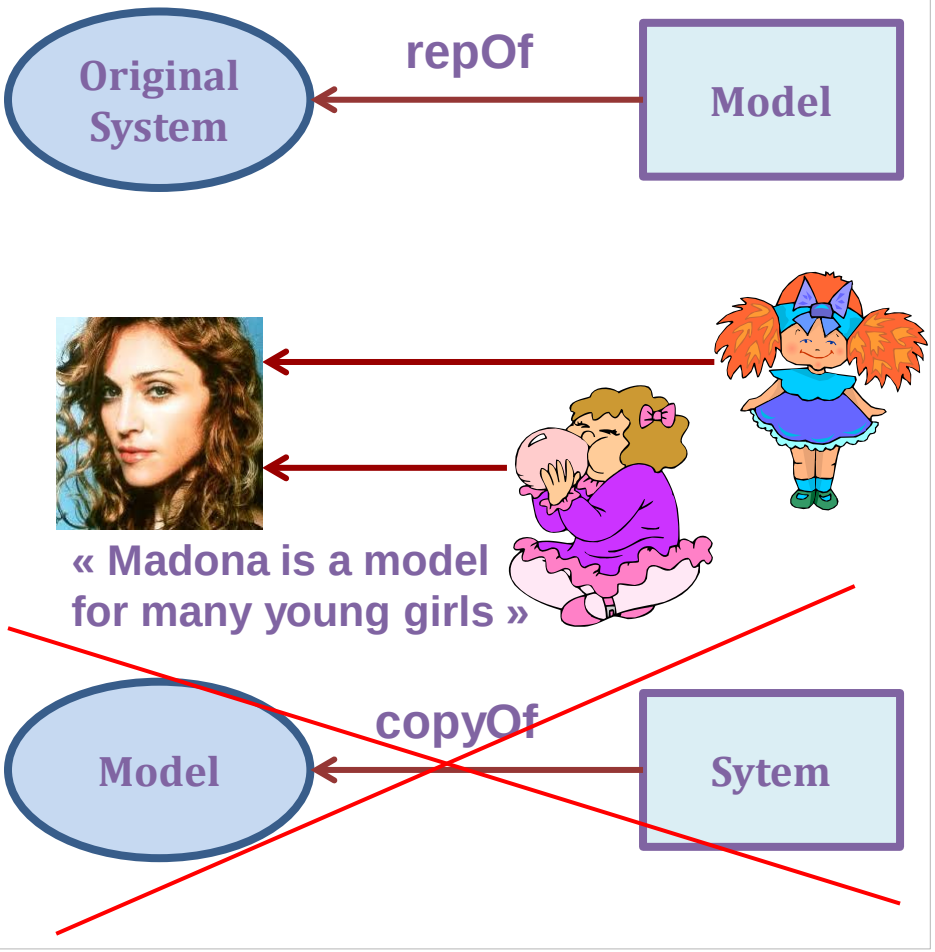
Summary of previous lectures

- Basic question
  - What is a model?
- We already know, in our context, that a MDE-model:
  - is a representation of a system
  - is written in the language of its metamodel
  - is a constrained directed graph
  - relies on basic definitions (graph representation)
  - if some of these definitions are changed, different TS
- What are the various kinds of MDE-models?
  - Unification/Classification



Beware: the word “model” is an “antagonym”

- A word that can mean the opposite of itself is an antagonym.
  - bound (bound for Tokyo, moving)
    bound (tied up, unable to move)
  - cleave (to cut apart)
    cleave (to seal together)
  - clip (attach to)
    clip (cut off from)
  - left (remaining)
    left (having gone)
- Model
  - [1] Copy of an original object
  - [3] Something that is copied
  - [6] an excellent example that deserves to be imitated



Most general definition of models

mod·el [módd'l] noun (plural **mod·els**)

1. copy of an object: a copy of an object, especially one made on a smaller scale than the original (*often used before a noun*)

2. particular version of manufactured article: a particular version of a manufactured article
had traded in her car for the latest model

3. something copied: something that is copied or used as the basis for a related idea, process, or system

4. somebody paid to wear clothes: somebody who is paid to wear clothes and demonstrate merchandise as a profession, for example, in fashion shows and photographs for magazines and catalogues

5. simplified version: a simplified version of something complex used, for example, to analyze and solve problems or make predictions *a financial model*

6. perfect example: an excellent example that deserves to be imitated

7. artist's subject: somebody who poses for a painter, sculptor, photographer, or other artist

8. zoology **animal copied by another animal:** an animal species repellent to predators which another animal mimics for protection

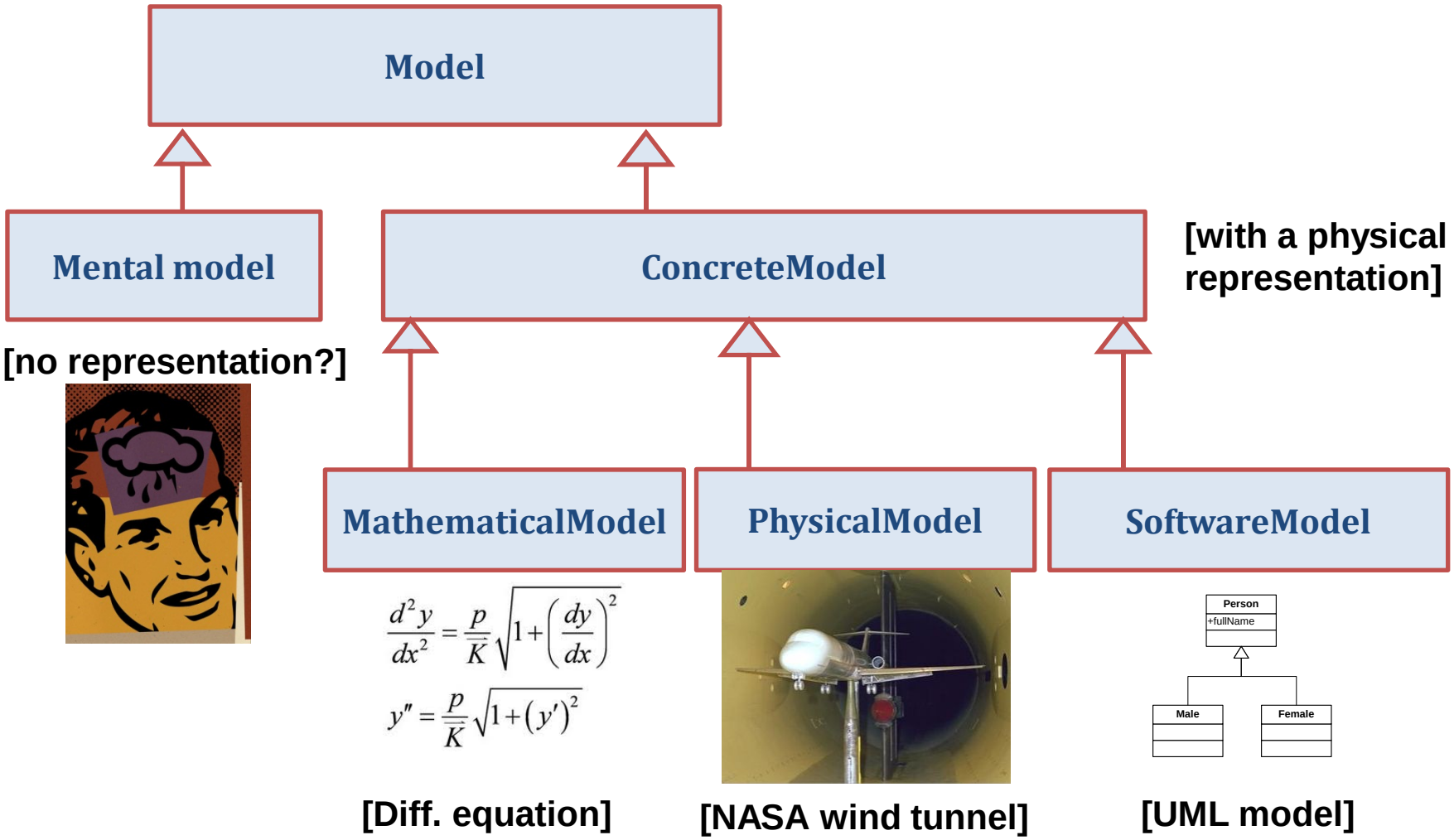
9. logic **interpretation:** an interpretation of a theory arrived at by assigning referents in such a way as to make the theory true

10. *U.K.* fashion **exclusive garment:** the first sewn example of a couturier's or clothing manufacturer's design, from which a new line of garments is produced

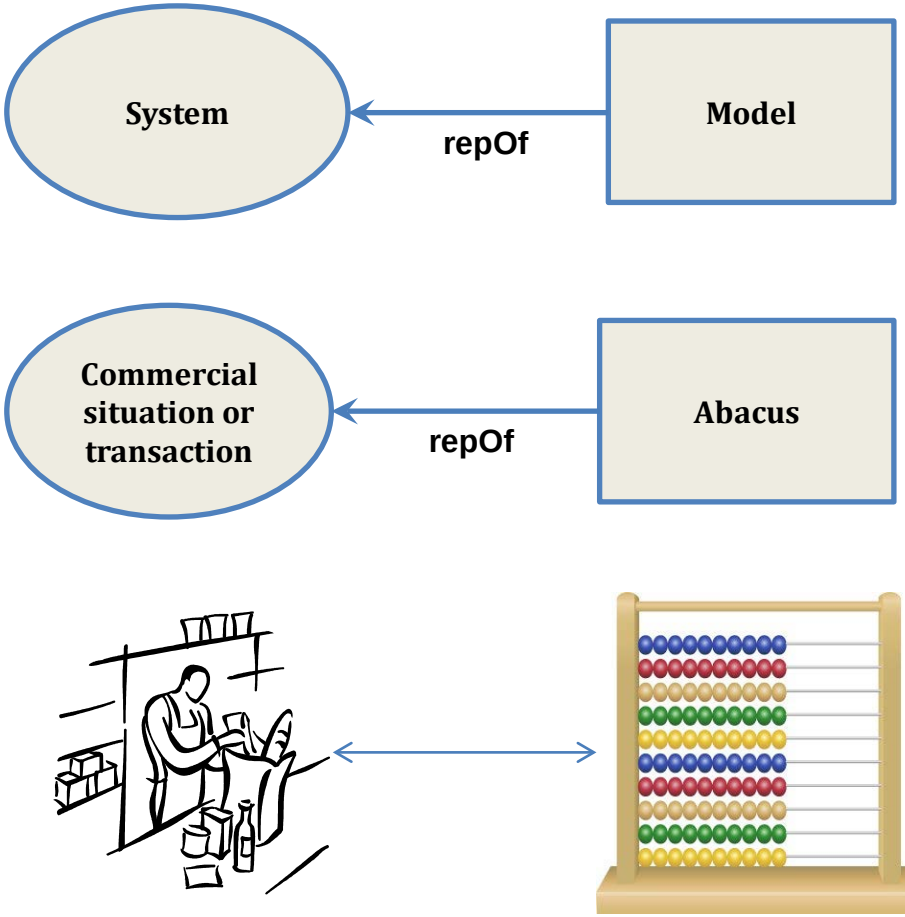
Models in the engineering field

- Our purpose is not to discuss models in philosophical terms
  - Our purpose is to use models for engineering
- More precisely to define an engineering based on models (MDE) that could be applied to several different fields (software engineering, data engineering, system engineering, business engineering)
  - As a consequence, our initial definition of models (general models) is more limited.

Ubiquitous Models



An example of a dynamic model (non software model)

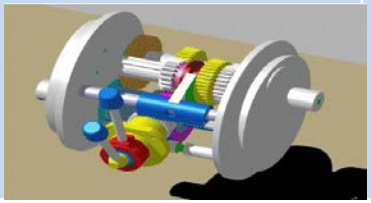


- An example of a physical dynamic system, made of wood, iron and plastic.
- Operations on the system translate into operations applied to the model.
- Requests on the system (questions) translate into requests on the model.
- In this case, the model is kept synchronized with the system.

About «software» models

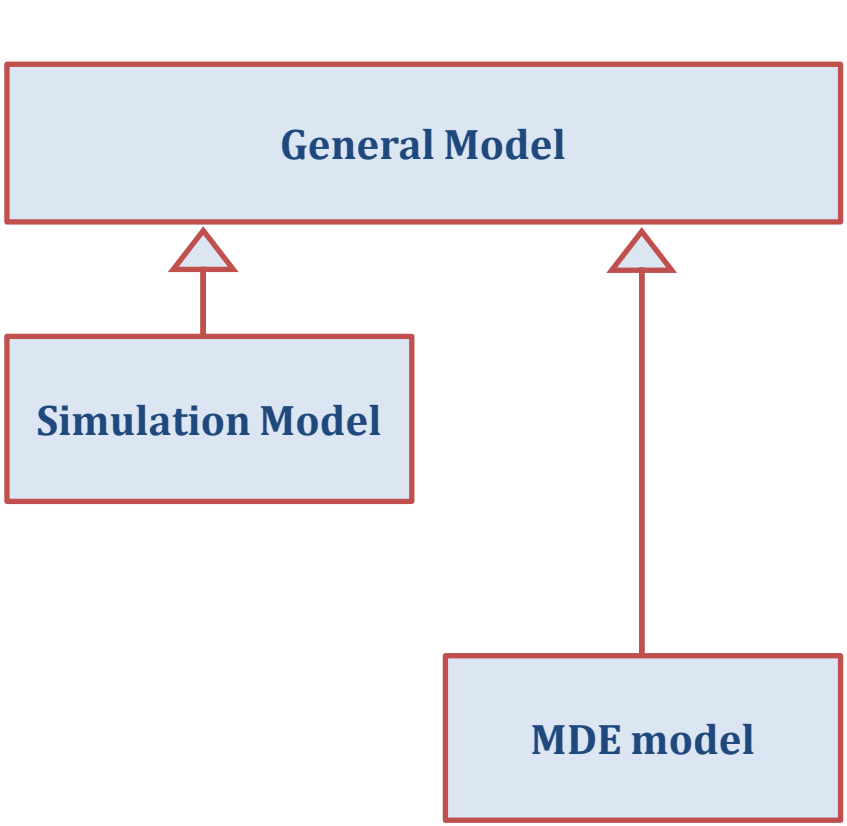
- What is a software model?
  - Ambiguous question
    - A model made of software?
    - A model of the software?
- Two different situations
  - Using software to manage any kind of models
  - Using models to manage software production, maintenance and operation
- Assuming situation (D)
  - S. Model Driven Software Engineering
  - S. Model Driven System Engineering
  - S. Model Driven Data Engineering
  - S. Model Driven Business Engineering

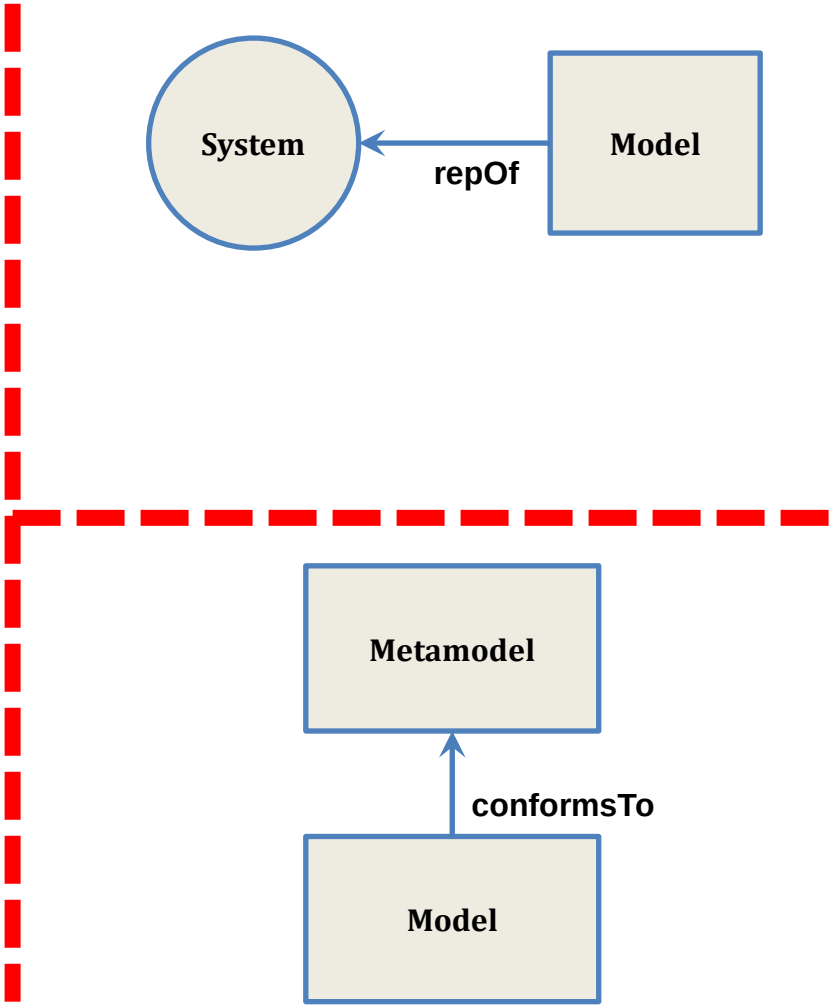
4 main posibillities

Support \ Goal	**Non software**	**Software**
Non software	**[A]** Airplane at the 1:20 scale	**[B]** No example(*)
Software	**[C]** ex. CATIA 	**[D]** Usual situation

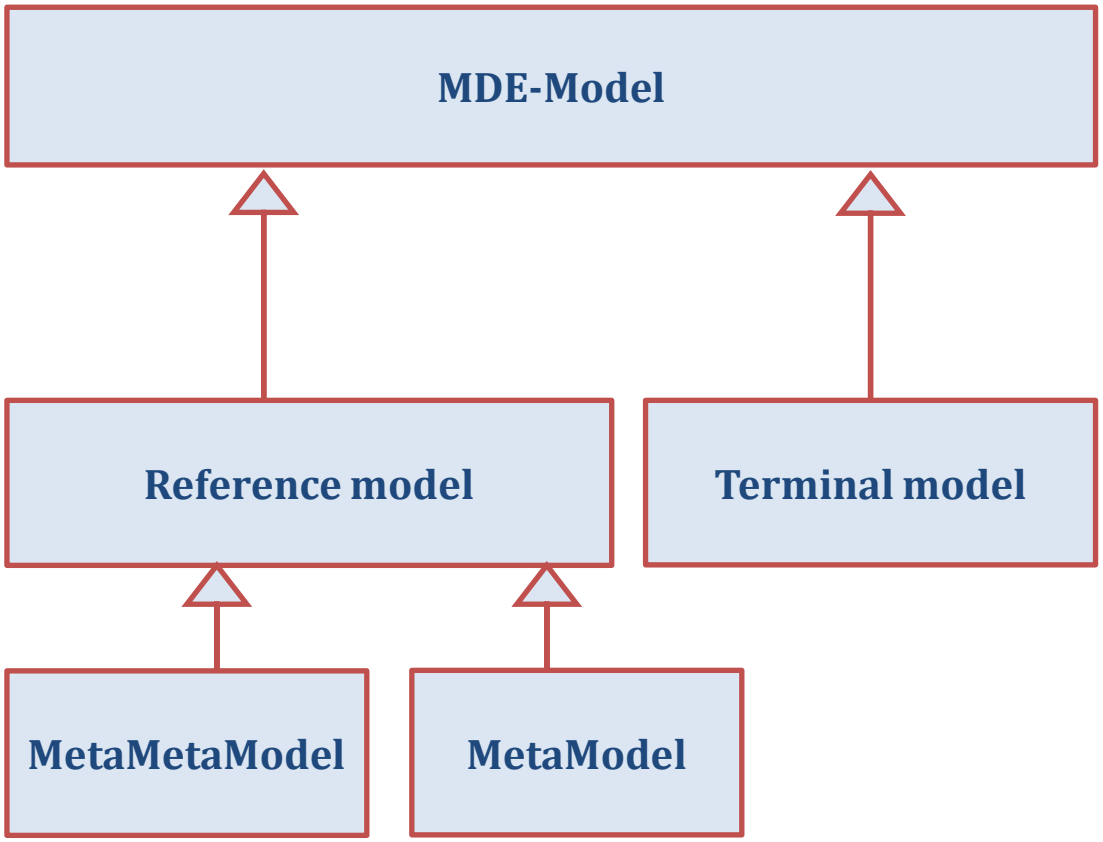
(*) See however «Software as cities» by Michele Lanza

Classification of software models (models made of software)





Classification of MDE-models



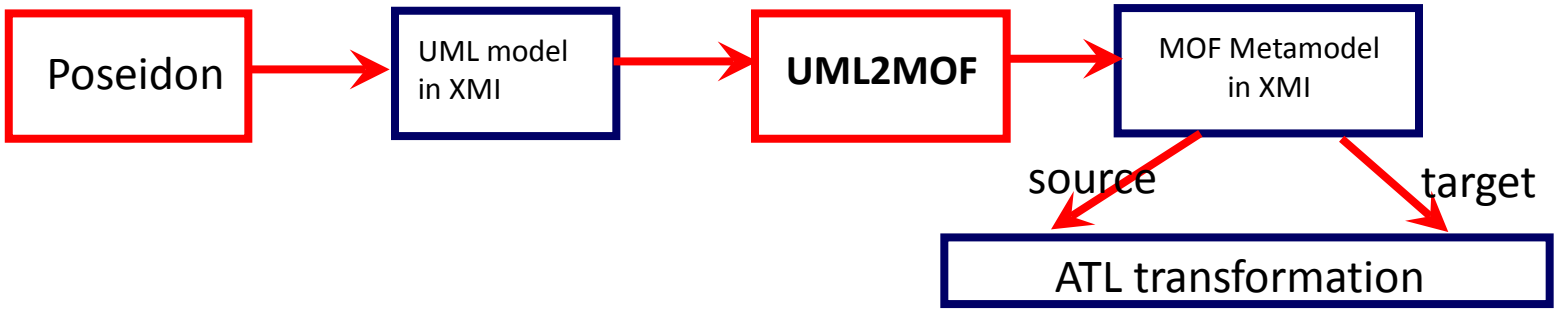
Many basic operations defined at the most general MDE-model level:

- ✓ Store in repository
- ✓ Retrieve from repository
- ✓ Transform in another rmodel
- ✓ etc.

Apply at all the derived levels

Metamodels as models

- Promotion and demotion
  - Promote: model -> referenceModel
  - Demote: referenceModel -> model
- Practical application
  - Early implementations
    - ✓ UML2MOF in MDR(*)
  - Promotions and demotions represented by transformations



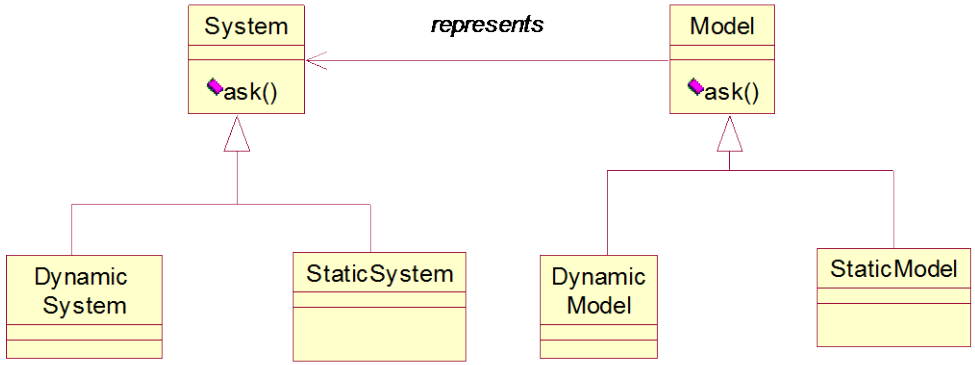
MDR(*) = SUN NetBeans MetaData Repository

Dual Models

- Product vs. Process
- Code vs. data
- Problem vs. Solution
- Static vs. Dynamic
- PIM vs. PSM (and CIM)
- Primitive vs. Derived
- Executable vs. Non executable
- Proprietary vs. Normative
- Atomic vs. Composite
- Basic vs. Correspondence
- Descriptive vs. Prescriptive (previously discussed)
- Formal vs. Informal (non relevant)

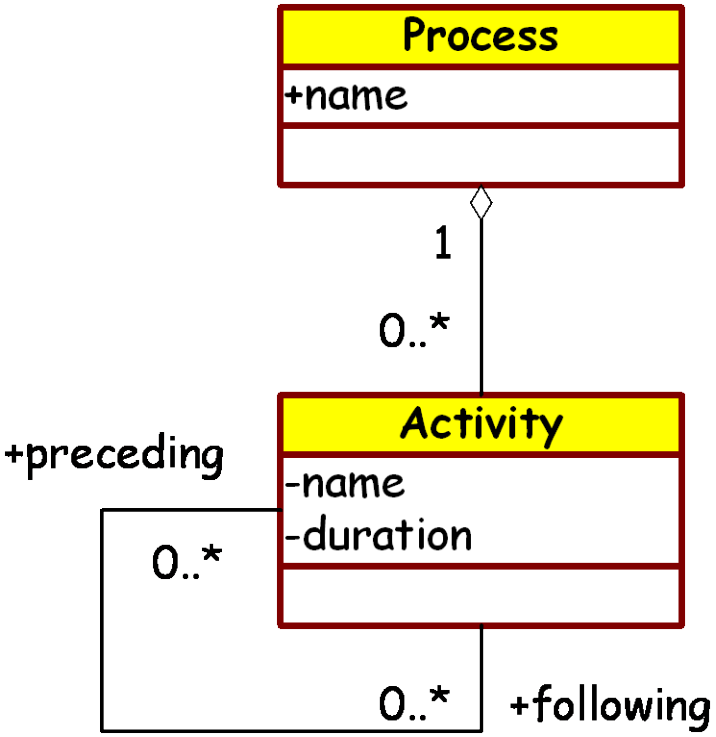
Static and Dynamic Systems/Models

- Most systems are dynamic
  - They evolve in time
  - Example : a washing machine
- Most models are static
  - They don't evolve in time
  - Example: a statechart of a washing machine
- Counter examples (rare)
  - Static system : Census results
  - Dynamic model : Simulation program

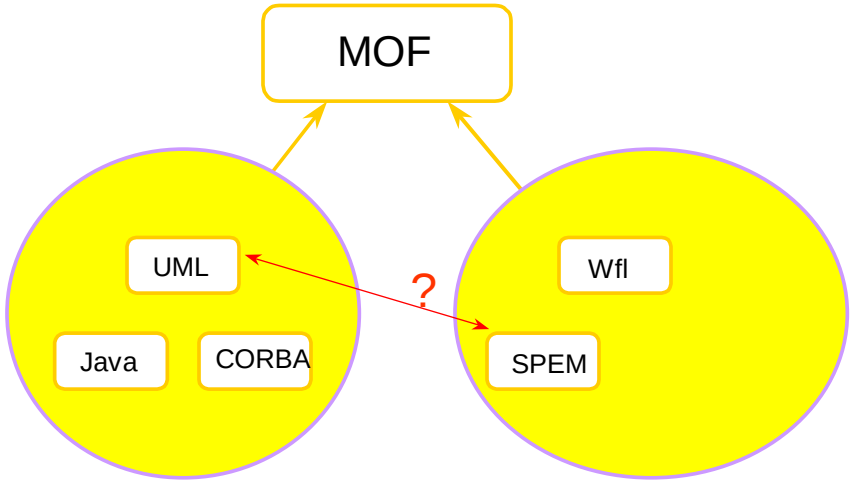


System \ Model	Static	Dynamic
Static	Rare	Impossible
Dynamic	Frequent	Simulation

Process and Product Models



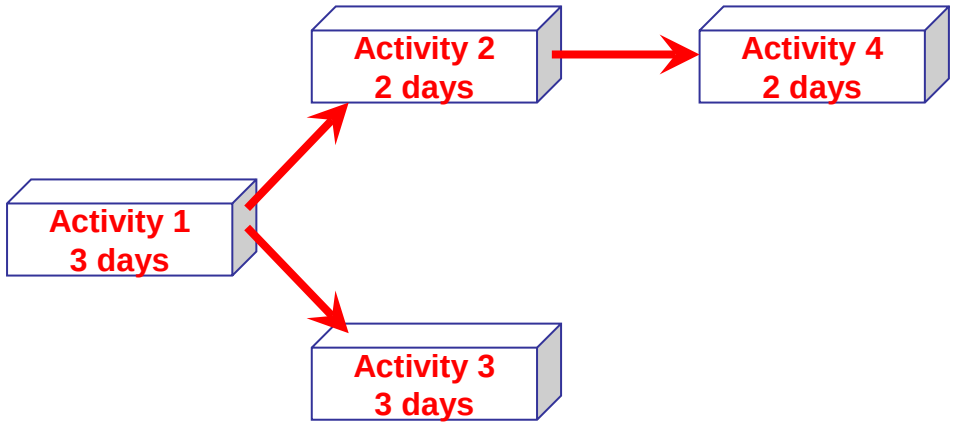
Product and process explicit models

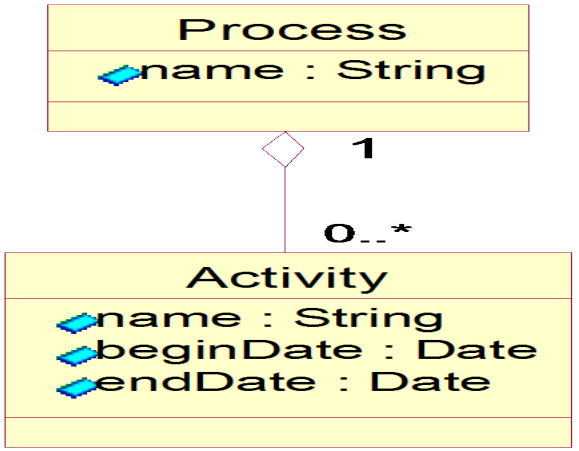


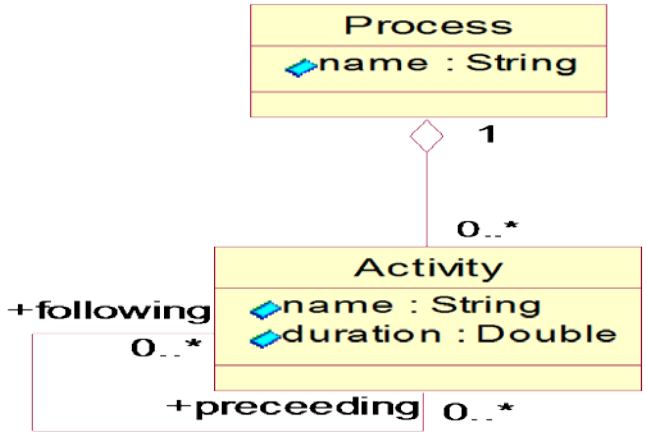
Who's doing what, when, how and why?

Gantt charts

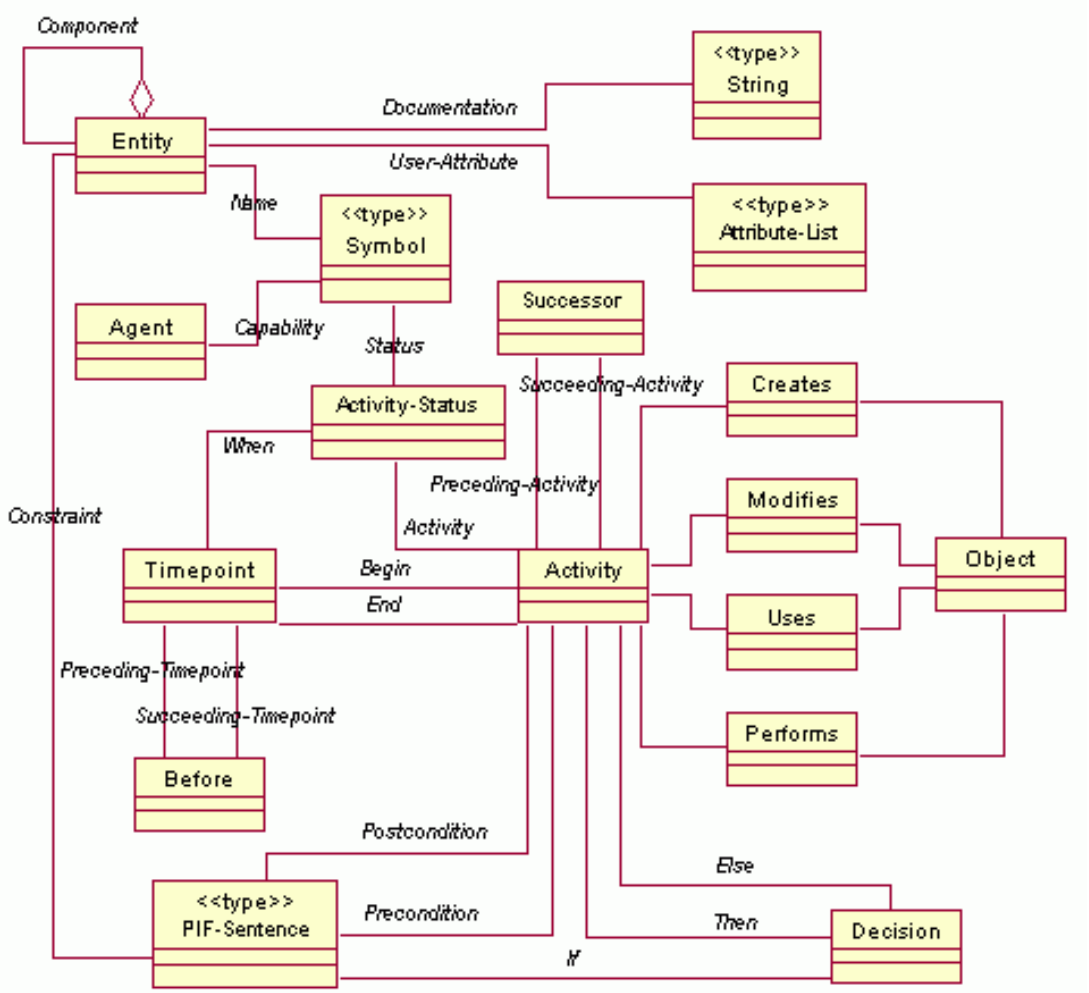
	01/01	02/01	03/01	04/01	05/01	06/01	07/01
Activity 1							
Activity 2							
Activity 3							
Activity 4							







Process Interchange Format and Framework

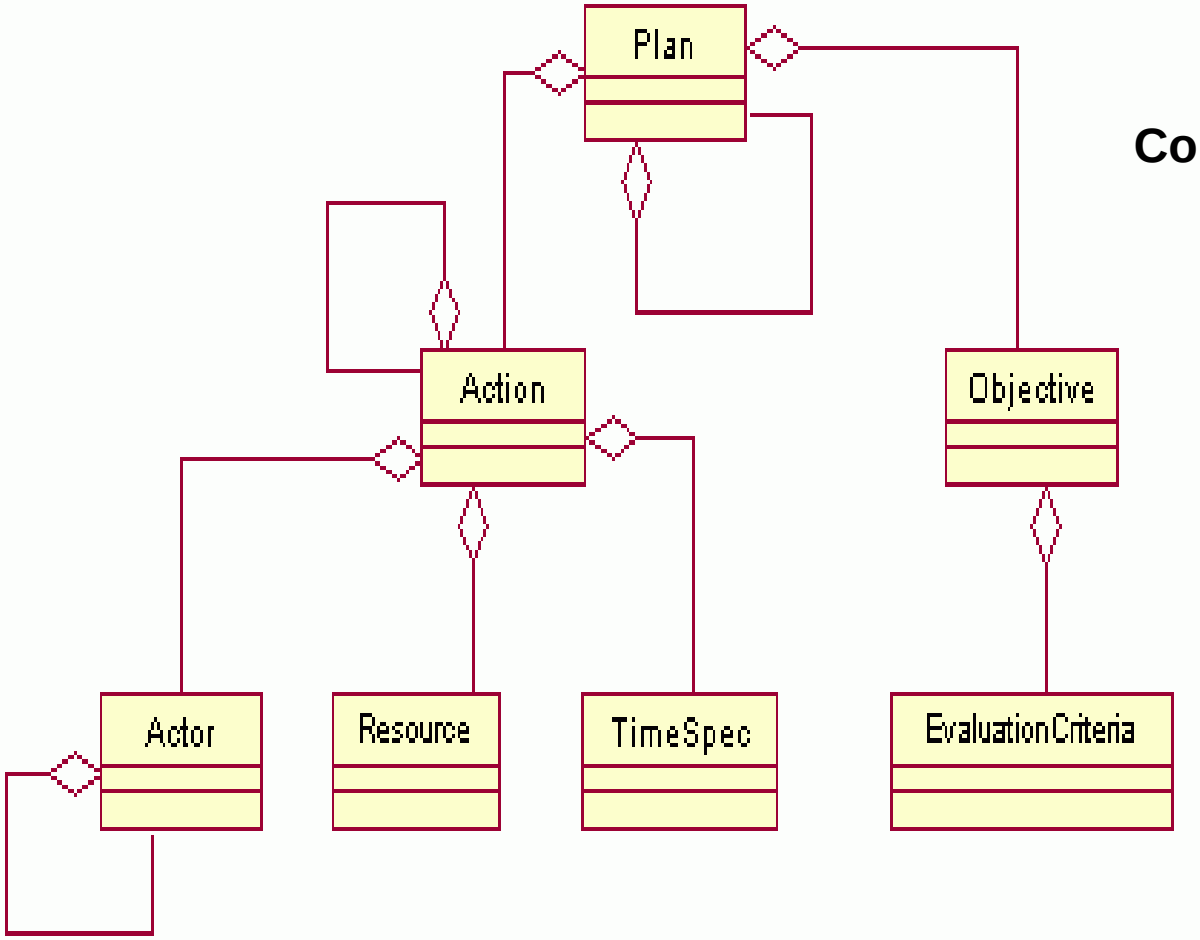


Process Interchange Format and Framework

PIF Core

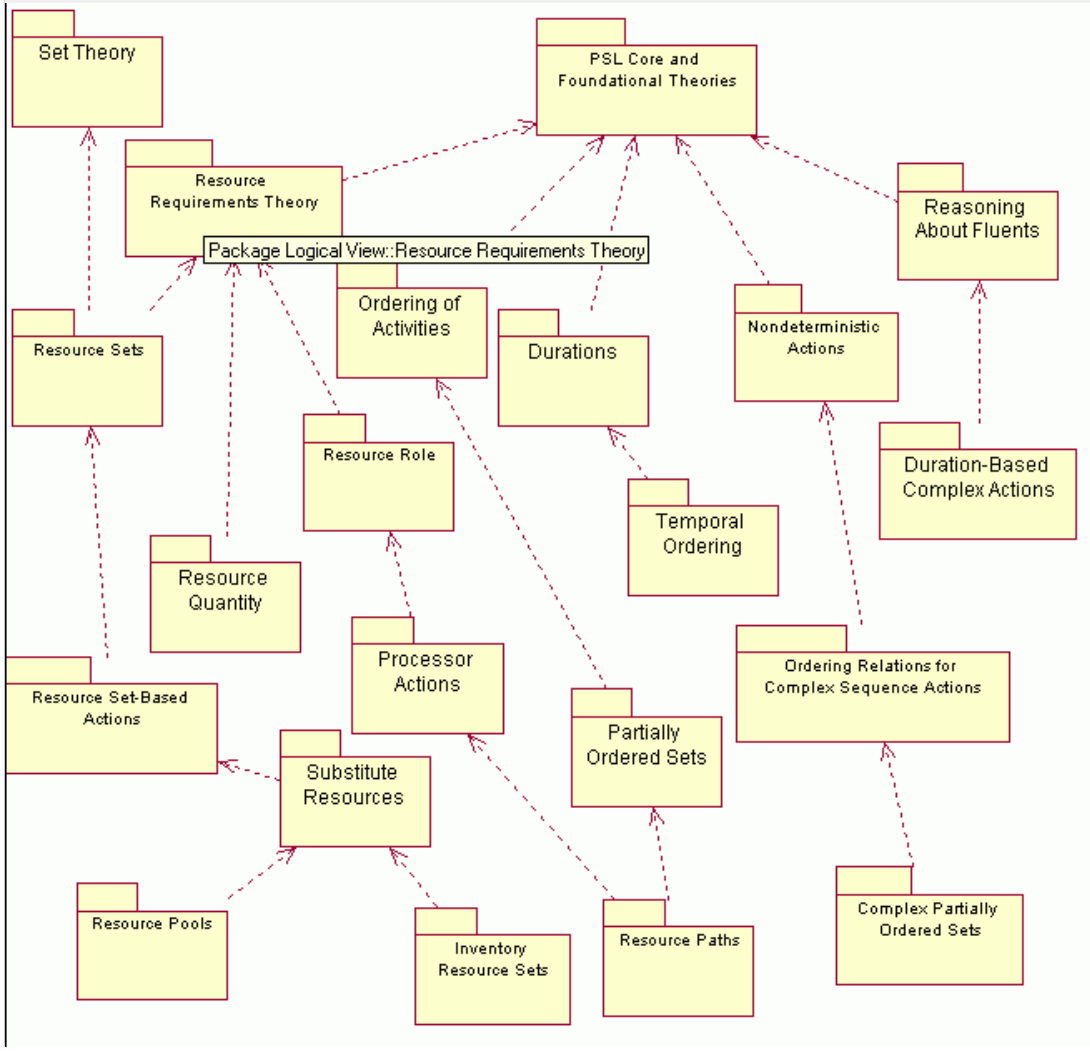
A process model may be enacted (on a workflow engine for example)

CPR (Core Plan Representation)



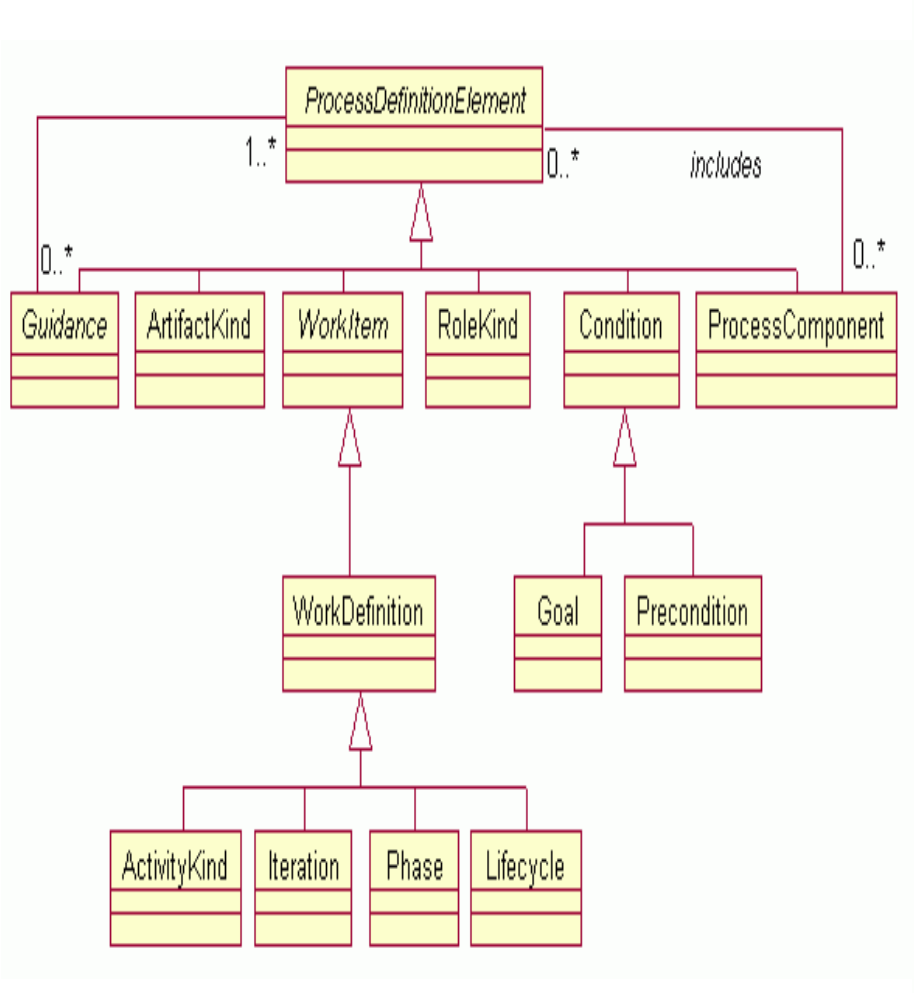
Core Plan Representation (DARPA)

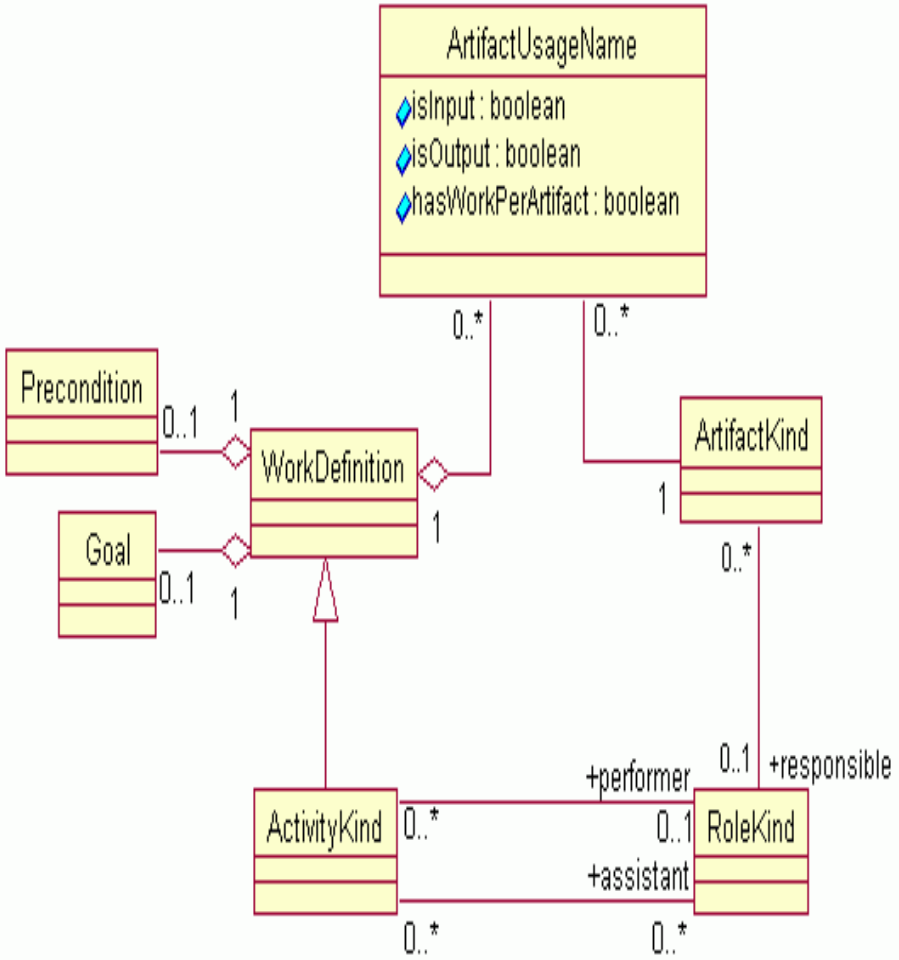
PSL (Process Specification Language)



Process Specification Language
NIST

Early SPEM (UPM)





Product and Process Patterns

- In many situations, modeling is conducted in two parallel branches, the product branch and the process branch:
  - Actigrams and datagrams in SADT
  - Product design and Process design in a building a factory
  - UML and SPEM at OMG
  - etc.
- The fact that a common representation exists for these to branches, with different metamodels, allows new possibilities of establishing synergy between the two branches.
- Patterns are emerging at this level of representation (P&P patterns)

Code and Data Models

- From code to data models
  - Initially MDE focused on code, for example code generation from abstract models
  - As it evolves, the subject of MDE is rapidly moving towards data
- Various reasons for this
  - Many model operations are essentially data transformation operations
  - When code is involved, it is usually considered as a special case of data
  - Most of MDE principles can readily be applied to data, with a gain in generalization
- This trend is accelerating
  - Important considerations about data may be considered from the modeling view
  - DataViz, OpenData, BigData, LinkedData, DataScience, etc.
  - New challenges ahead (scaling up)

Data transformation

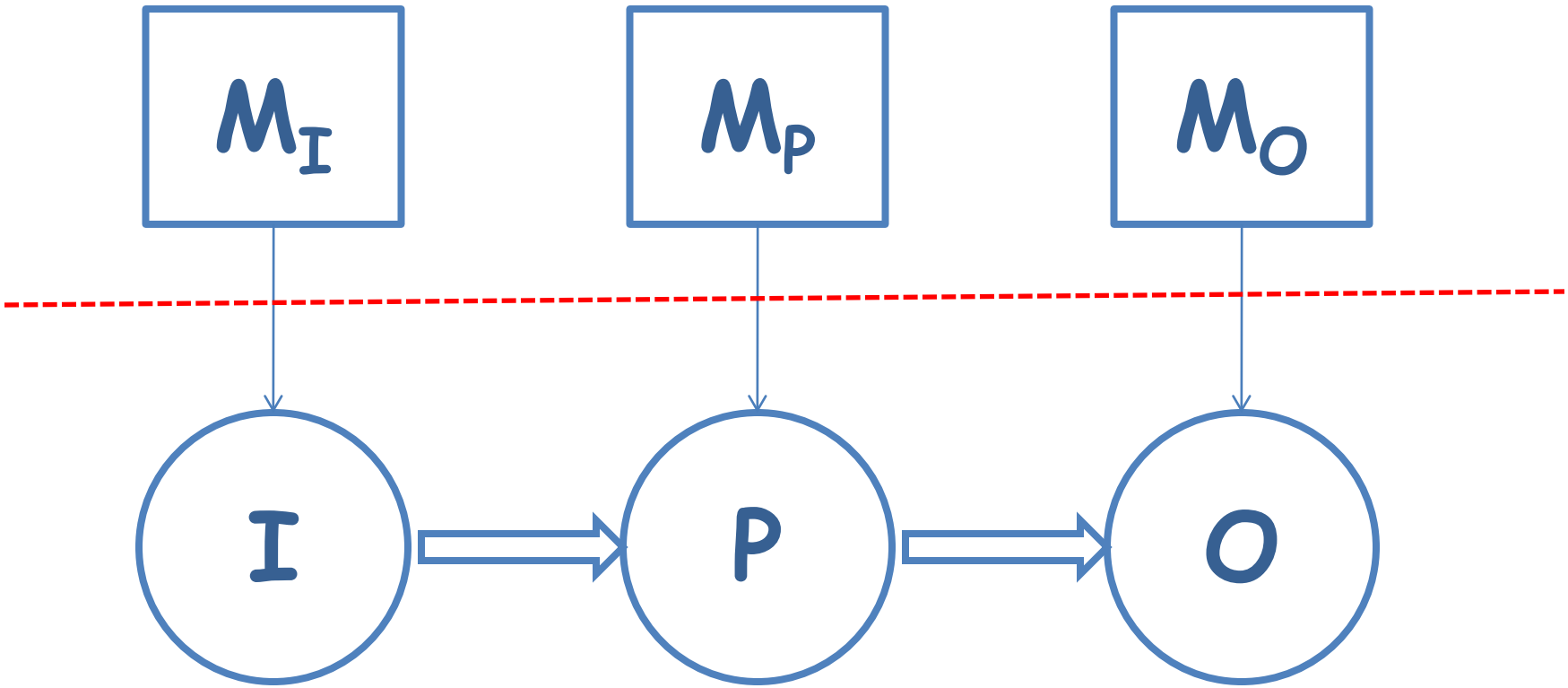
A broad spectrum of data is available on the Web in distinct heterogeneous sources, stored under different formats. As the number of systems that utilize this data grows, the importance of data conversion mechanisms increases greatly… A key observation is that, often, the application programs used by organizations can only handle data of a specific format... To enable a specific tool to manipulate data coming from various sources, a translation phase must take place…The naive way to translate data from one format to another is writing a specific program for each translation task… A sound solution for a data integration task requires a clean abstraction of the different formats in which data are stored, and means for specifying the correspondences between data in different worlds and for translating data from one world to another.

Serge Abiteboul & al.
Tools for data translation and integration
Data engineering Vol. 22, N.1, march 1999.

Sensor Data Stream Processing

- Many systems are becoming world-wide data-centric and transformation-based processing networks.
- The topology of the network is constantly evolving with new data sources constantly appearing or disappearing, new data translation, merging, control or measure being added, deleted or updated, etc.
- The distance between the creation of the data and its final use is increasing.
- Sensors are no more producing streams of bits or bytes, but streams of intelligent data with precise semantic.
- Each time the software or driver of a sensor has a version change, the data semantics may evolve.
- Managing such complex systems necessitates the ability to deal with the changing metadata of the various source, targets or intermediary processing nodes in the network and keeping trace of the metadata including provenance information.

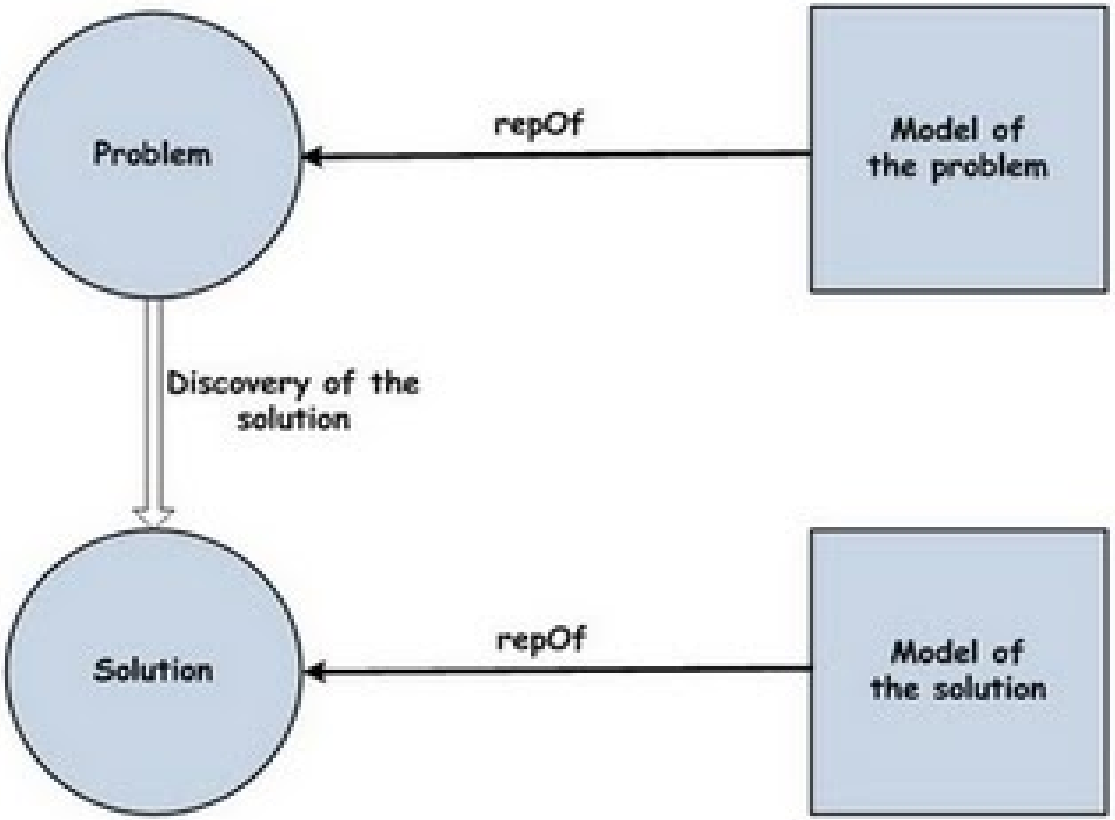
Code and Data Models
Uniform DSL representation



Problem and Solution Models

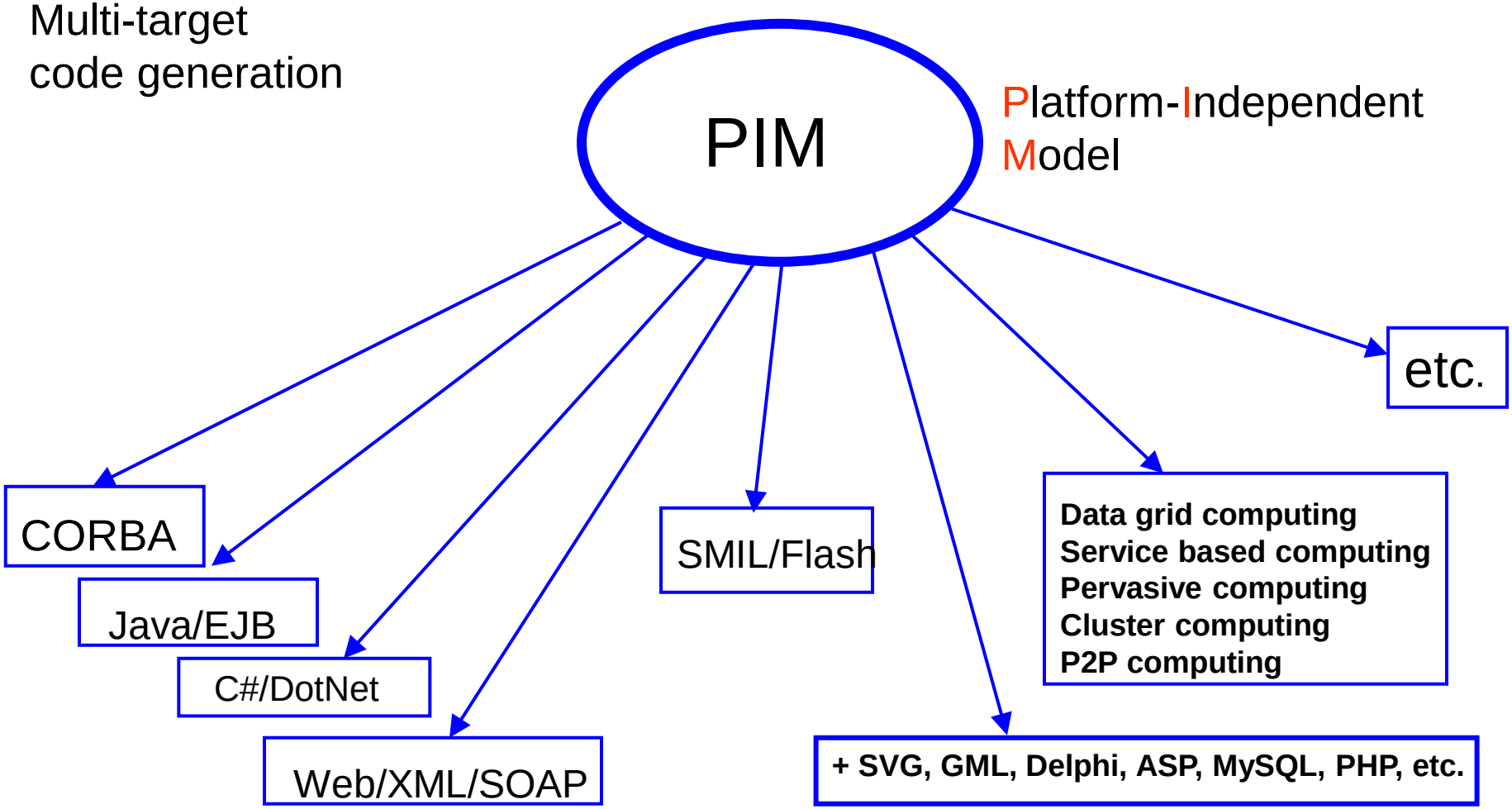
- Once upon a time, there was a team leader that was going on holidays. Before leaving, (s)he made the last recommendation to his/her small team of three young engineers:
  - For the ongoing project, do not start coding in Java before the UML model is completely finished and that you all agree on this model.
- On the Monday morning, as soon a s/he left, one of the engineers told the others of a wonderful discovery he made while twittering in the week end:
  - a very powerful tool to generate UML diagrams from Java code.
- The decision was rapidly taken and all three of them started coding the problem in Java.
- Some days before the end of the holidays of their leader, all the Java code was used to generate UML diagrams and both the code and the UML diagrams were handled to the group leader.
- S/He was quite impressed at the level of detail of the UML model and the narrow correspondence between the code and the model.

Problem and Solution Models

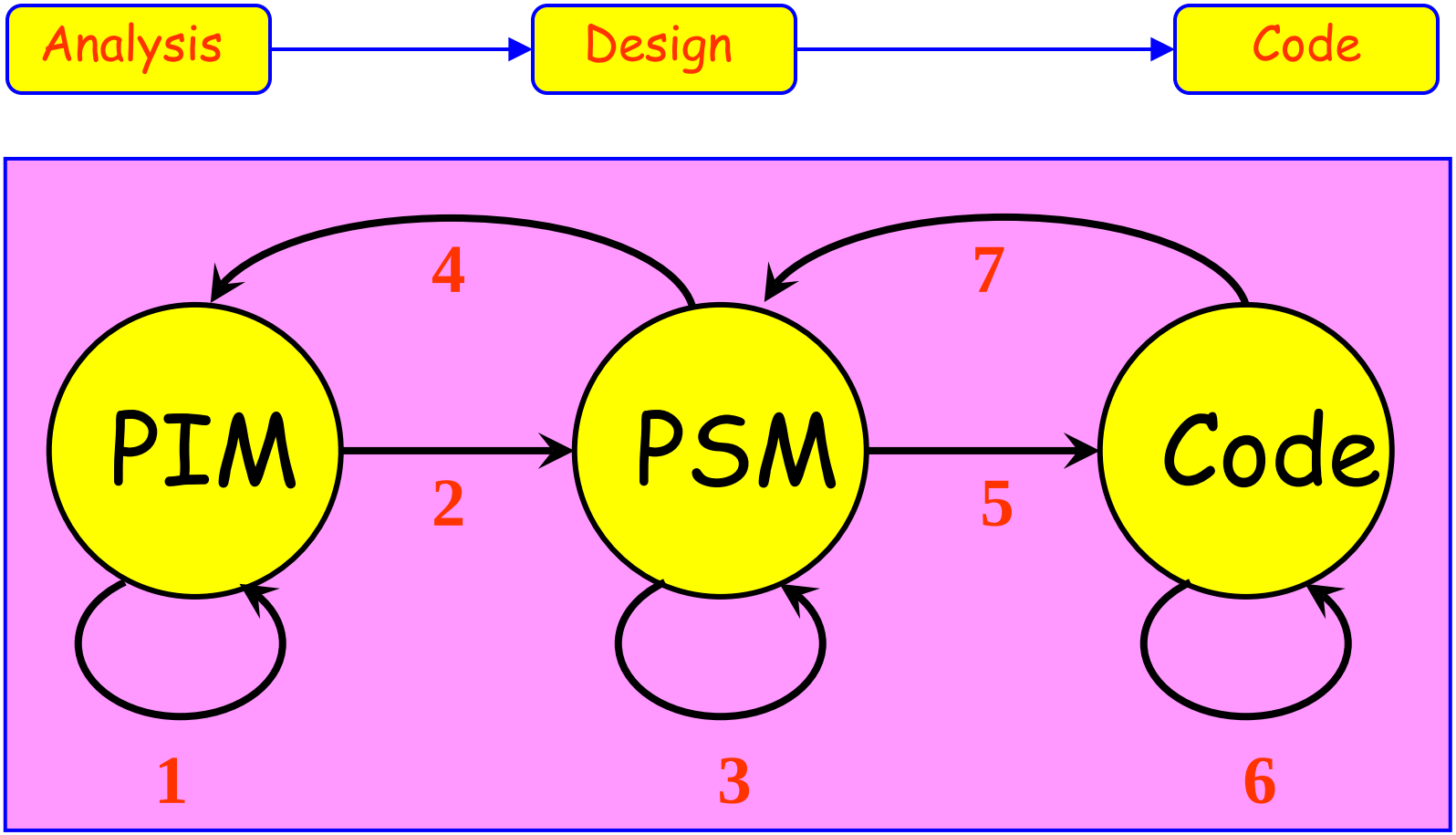


*MDE is a unique chance to achieve the goal of separation of the **what** and of the **how**, for example in the educational context.*

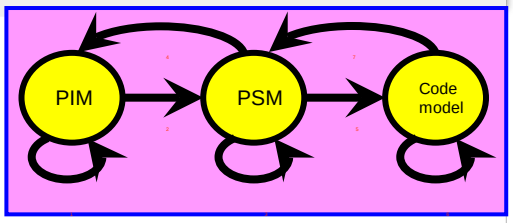
PIM to PSM

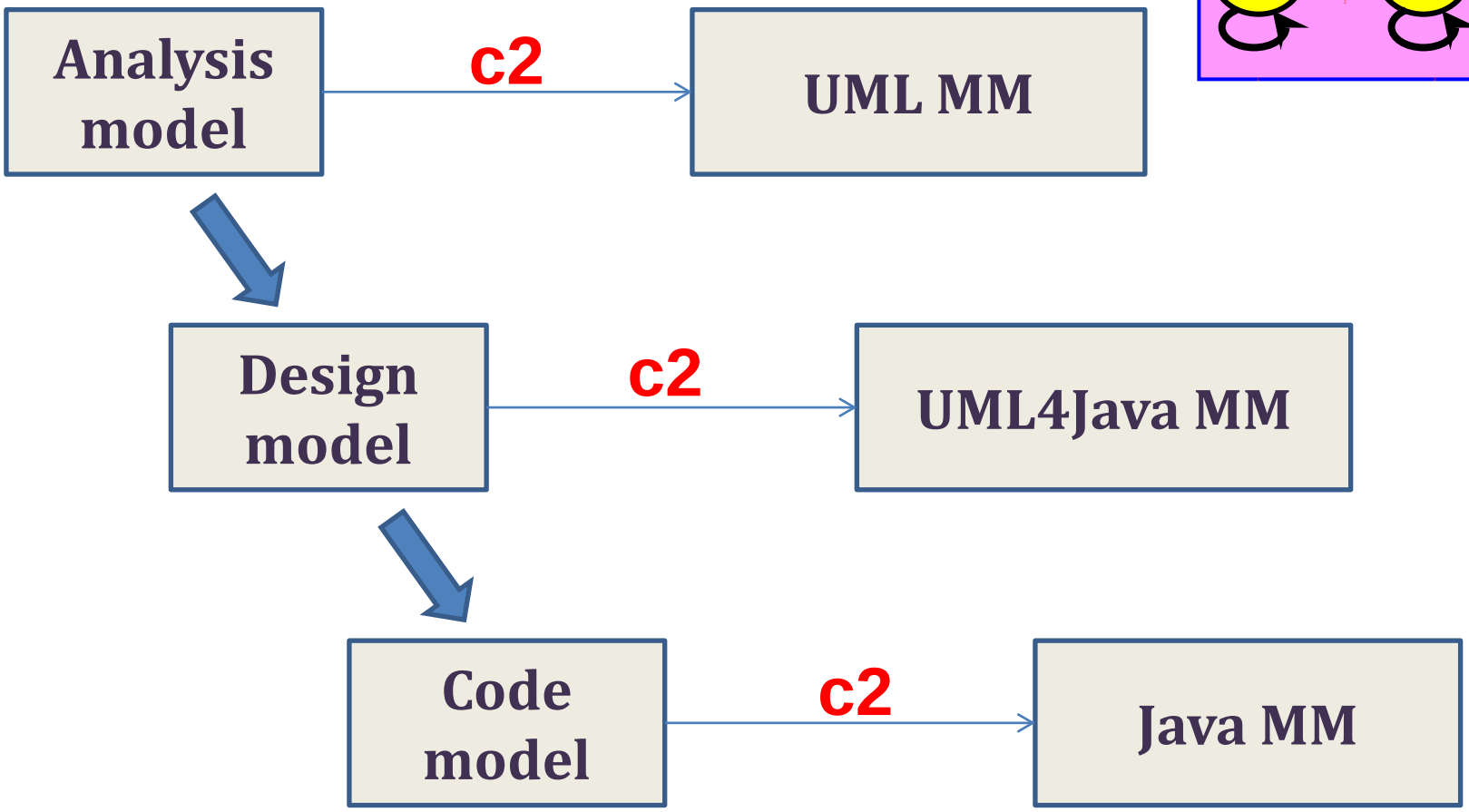

Multi-target
code generation
PIM
Platform-Independent
Model
etc.
CORBA
Java/EJB
C#/DotNet
Web/XML/SOAP
SMIL/Flash
Data grid computing
Service based computing
Pervasive computing
Cluster computing
P2P computing
+ SVG, GML, Delphi, ASP, MySQL, PHP, etc.

Development cycle: old and new

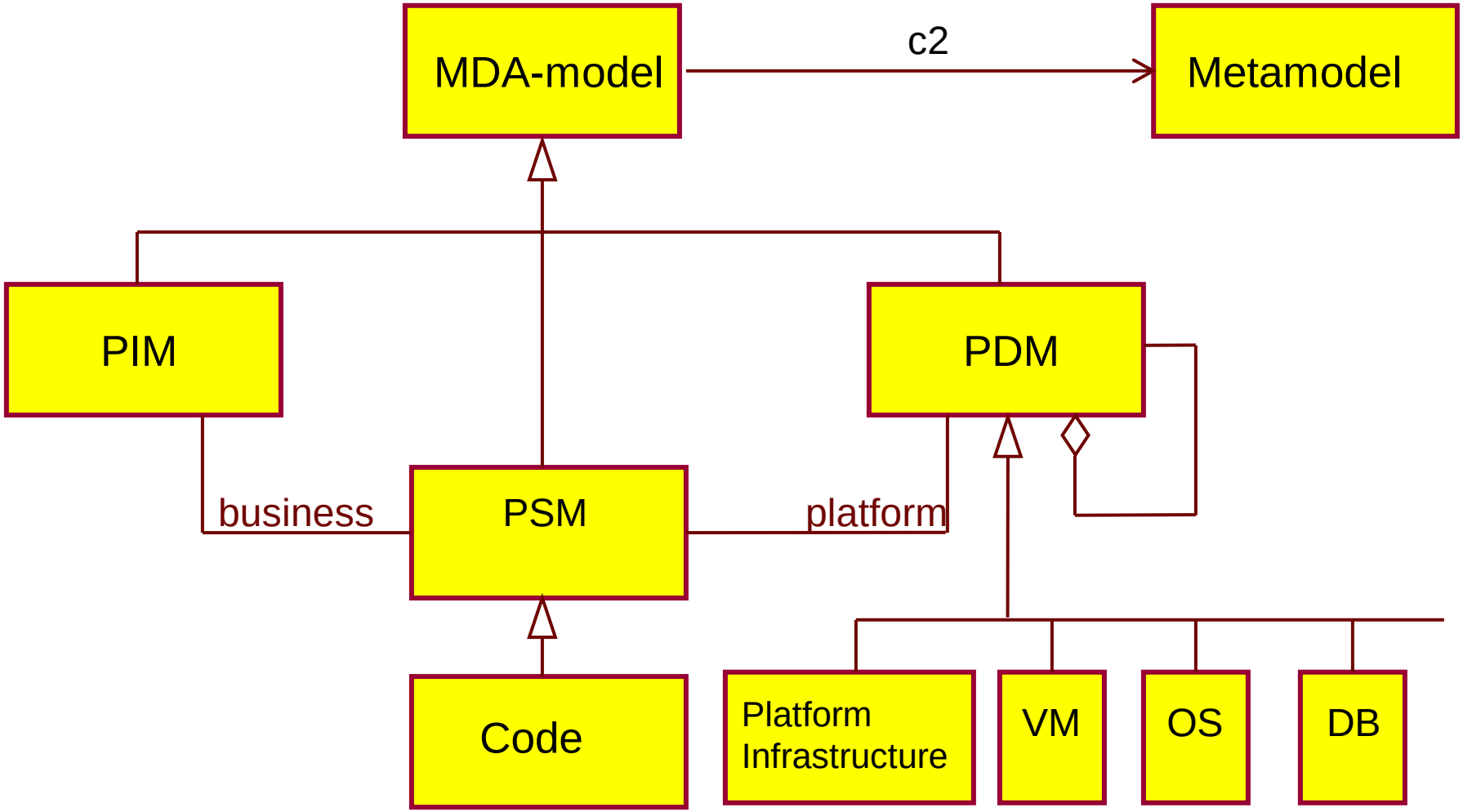


Possible MDA organization





Some OMG MDA models



Platforms as models?

- Difficulty to define platform specific models (PSMs) without having defined precisely what a platform is

MDA Guide (Draft Version 0.2)

Document Number: ab/2003-01-03
Date: 23 January, 2003

Copyright © 2003 OMG

Editors: Joaquin Miller and Jishnu Mukerji.

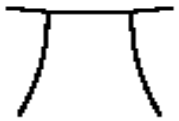


Figure 2-1 A platform

Many of the illustrations in this Guide use this icon to represent a platform.

2.2.7 Platform

Examples:

Generic platform types

Object: A platform that supports the familiar architectural style of objects with interfaces, individual requests for services, performance of services in response to those requests, and replies to the requests. [5]

Batch: A platform that supports a series of independent programs that each run to completion before the next starts.

Dataflow: A platform that supports a continuous flow of data between software parts.

Technology specific platform types

CORBA: An object platform that enables the remote invocation and event architectural styles. [formal-01-12-35]

CORBA Components: An object platform that enables a components and containers architectural style. [formal- - -]

Java 2 Components: Another platform that enables a components and containers style.

Vendor specific platform types

CORBA: Iona Orbix, Borland VisiBroker, and many others

Java 2 Components: BEA WebLogic Server, IBM WebSphere software platform, and many others

Microsoft .NET

Executable and non-executable models

- Most models are not directly executable
  - Some models are executable
  - For a model to be executable, we usually need:
    - ✓ an action language (often textual)
    - ✓ a graph visiting strategy
  - Many models non natively executable may be translated to executable code
  - A weaker property is animation. Some models may be animated often for the purpose of illustration
  - As seen before, a related property is enactment, only for process models (≠ executable models)
- Current efforts on executable UML at OMG

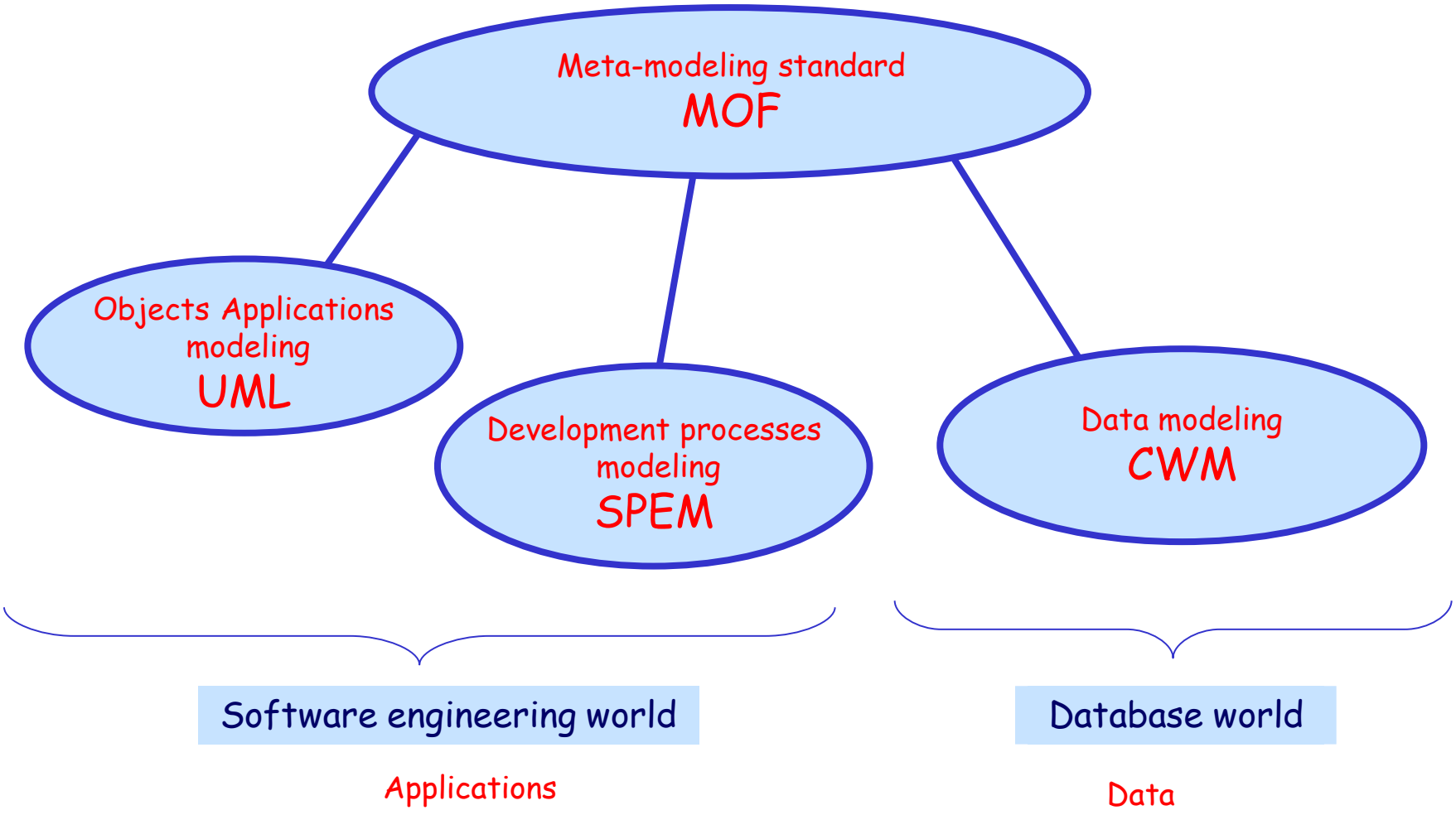
Proprietary vs. standards metamodels

- Like an ontology, a metamodel defines a consensual shared abstraction of a given system (situation, phenomena, pattern, etc.).
- As such it is intended to be used by groups of people (open or closed groups).
- The metamodel is likely to evolve, to follow the needs and collective decisions of the group
- The development and tuning of a metamodel may represent important costs. (A good metamodel does not come for free).
- One important open group producing recommendations in the form of metamodels is OMG:
  - The OMG does not directly produce tools corresponding to metamodels
  - The metamodels produced by OMG are informal documents or XMI files.
  - The verification of these metamodels is not usually automated
  - The version changes of these metamodels are not yet represented as concrete deltas

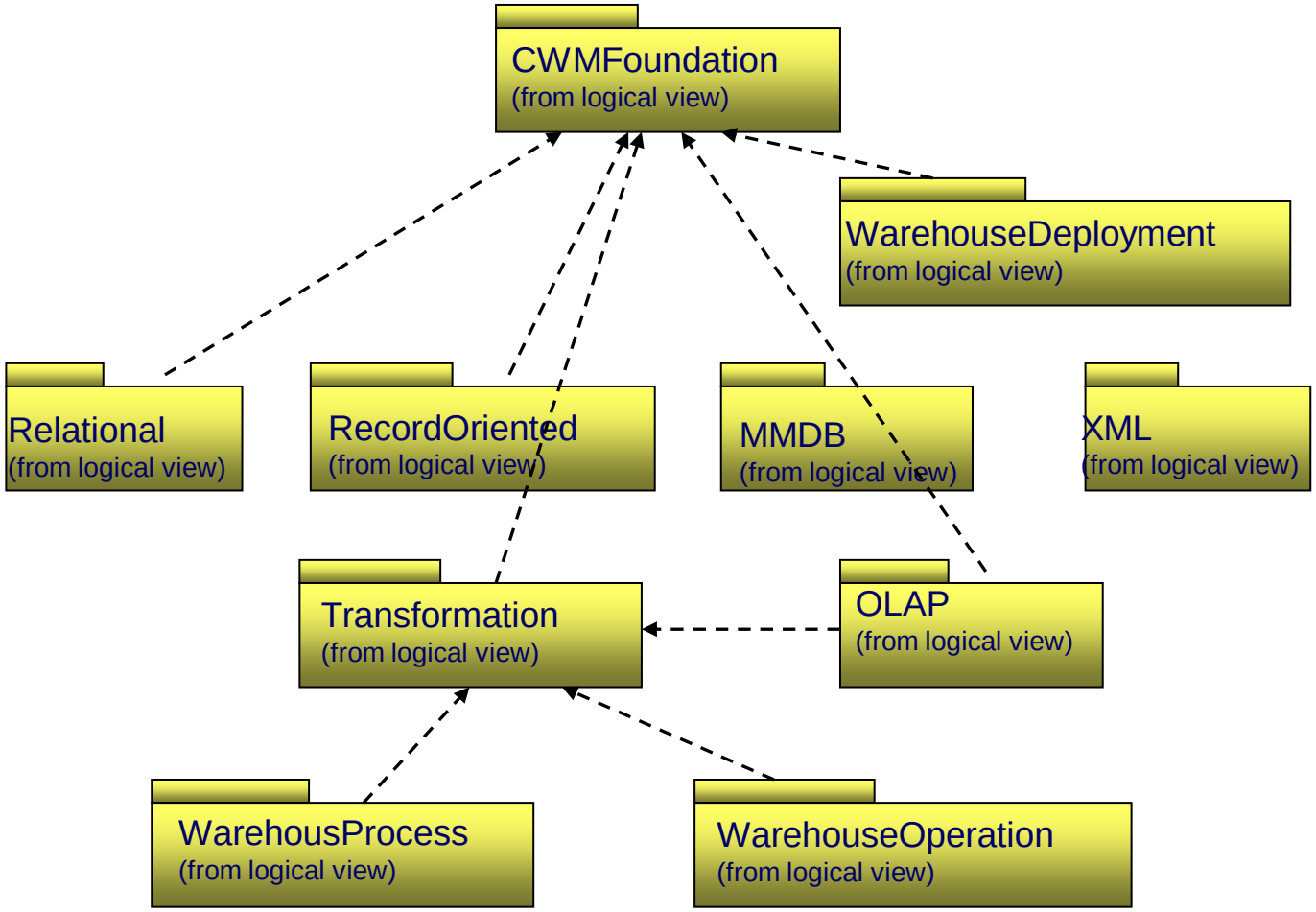
Some OMG MDA recommendations

Common Terminology Services 2 (CTS2)
Common Warehouse Metamodel (**CWM**™)
Common Warehouse Metamodel (CWM™) Metadata Interchange Patterns (MIP)
Concrete Syntax for UML Action Language [Action Language for Foundational UML] (**ALF**)
Diagram Definition (DD)
Meta Object Facility (**MOF**™)
Model Driven Message Interoperability (**MDMI**)
Model-level Testing and Debugging (MLTD)
MOF 2.0 Facility and Object Lifecycle (MOFFOL)
MOF Models to Text Transformation Language
MOF™ Query / View / Transformation (**QVT**)
MOF™ Support for Semantic Structures (**SMOF**)
MOF™ 2.0 Versioning and Development Lifecycle (MOFVD)
Object Constraint Language (**OCL**)
OMG Systems Modeling Language (**SysML**)
Ontology Definition Metamodel (**ODM**)
Reusable Asset Specification (**RAS**)
Semantics of a Foundational Subset for Executable UML Models (**FUML**)
Service oriented architecture Modeling Language (**SoaML**®)
Software Process Engineering Metamodel (**SPEM**)
Systems Modeling Language (**SysML**)
SysML-Modelica Transformation (SyM)
Unified Modeling Language™ (**UML**®)
UML Diagram Interchange (**UMLDI**)
UML Human-Usable Textual Notation (**HUTN**)
MOF 2 XMI Mapping (**XMI**®)

General view of OMG recommendations



CWM high level packages



XMI (XML Model Interchange)

- XMI enables to specify an accurate mapping from MOF to XML
  - Allowing the exchange of any MOF-compliant metamodel s and corresponding models
  - Allowing the DTD's and Schema automatic generation for any MOF-compliant meta-model
- Benefits
  - Model exchange (import/export)
  - No compatibility was previously existing between those formats
  - Based on OMG standards: XML et MOF (agreement OMG/W3C)
  - Designed to be transmitted as a dataflow, Platform independent
- Historical background
  - suggested as an answer to a RFP for a model exchange format (SMIF : Serial Model Interchange Format)
  - first proposition in June 1998, revision in November 1998
  - submitted for standardization in January 1999,
  - adopted in February 2000 (version 1.0)

UML Human-Usable Textual Notation (HUTN)

- UML Human-Usable Textual Notation (HUTN)
- A specification for a Human-Usable Textual Notation (HUTN)
  - for expressing other specifications
  - in terms of the UML Profile for Enterprise Distributed Computing (EDOC)
  - and its companion UML Profile for CORBA.
- HUTN offers three main benefits.
  - (1) It is a generic specification that can provide a concrete HUTN language for any MOF model;
  - (2) The HUTN languages can be fully automated for both production and parsing; and
  - (3) The HUTN languages are designed to conform to human-usability criteria.
- XMI difficult to read by human operators …
- … hence the idea to develop a textual language: HUTN
- Proposition done in 1999 by Data Access, DSTC, IBM, Open-IT and Unisys under the direction of OMG
- HUTN must be more "simple" than XMI, being user-oriented
- HUTN Main objective: getting closer to well-known languages (Java, C++…) in order to be easily understandable thus easily usable and easy to handle for a user

Java MetaData Interchange

JMI

JSR 40
The JavaTM Metadata Interface (JMI) Specification

The Java Metadata Interface specification will address the need for a pure Java metadata framework API that supports the creation, storage, retrieval, and interchange of metadata

- Just like XMI allows to define a matching between MOF and XML models, the JMI specification defines a matching between MOF and Java models.

- Many IDE's (Integrated Development Environment) are written in Java (Together, ArgoUML, Poseidon, NetBeans, EMF, etc.). JMI makes the access to MOF models easier from those environments.

- JMI can manage metadata warehouse, i.e. data from models and MOF metamodels

Enterprise Application Integration (EAI)

- UML Profile for EAI
- Enterprise Application Integration (EAI) is defined as the use of software and computer systems architectural principles to integrate a set of enterprise computer applications.
- The goal is to simplify application integration by standardizing application metadata for invoking and translating application information.

- The boundaries of this profile are defined by three scenarios representing the integration needs
  - Applications integration by connectivity
  - Applications integration by information sharing
  - Applications integration by processes collaboration

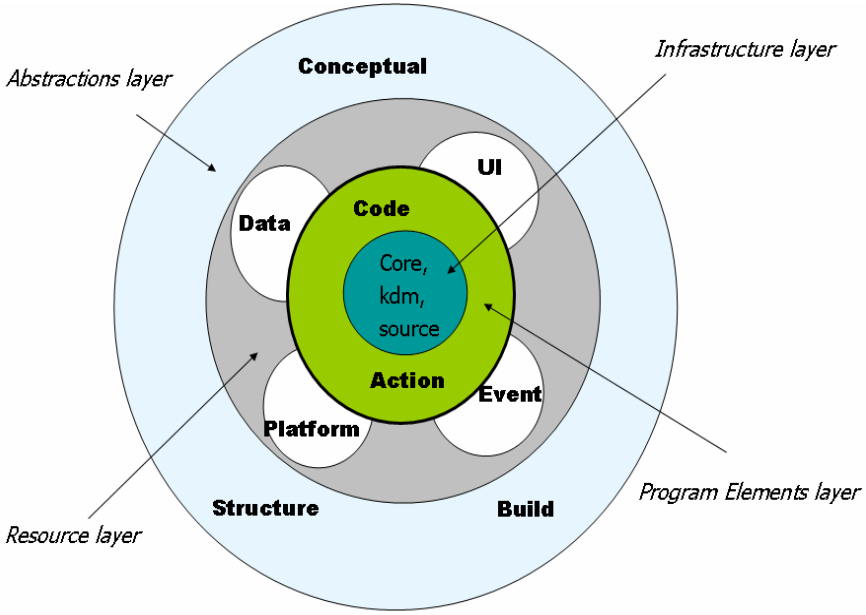
Enterprise Distributed Object Computing (EDOC)

- **UML Profile for Enterprise Distributed Object Computing (EDOC)**
- The vision of the EDOC Profile is to simplify the development of Component based EDOC systems by means of a modeling framework, based on UML and conforming to the OMG Model Driven Architecture.
- The EDOC profile aims at making enterprise, systems or organizations models (PIM's) development easier, using a recursive collaboration approach at several granularity and purely different coupling degrees levels.

- This profile may be useful in EAI, B2B, B2C, … applications. Complex relations between EAI and EDOC profiles.
- The objective is to integrate process and information models
- Dealing with business collaborations is important: alliances, outsourcing, supply chains, electronic business, etc.
- The business process engineering management is done via the services assembling, aiming at the global architecture stability in the local modifications context
- The EDOC profile owns an ECA (Enterprise Collaboration Architecture) PIM profile

KDM (Knowledge Discovery Metamodel)

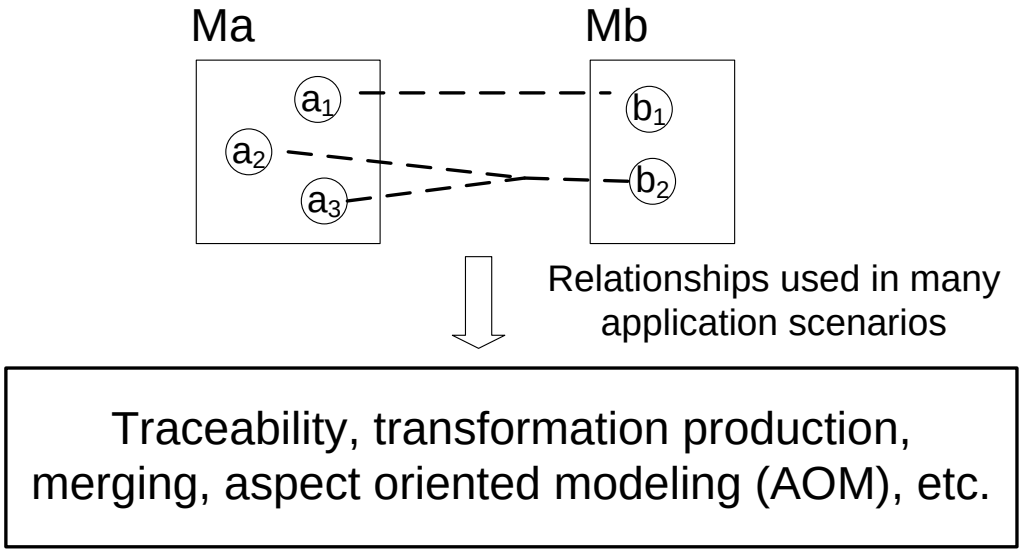
- The goal of KDM is to ensure interoperability between tools for maintenance, evolution, assessment and modernization.
- KDM is defined as a *metamodel* that can be also viewed as an *ontology* for describing the key aspects of knowledge related to the various facets of enterprise software.
- Support of the KDM means investment into the KDM ecosystem - a growing open-standard based cohesive community of tool vendors, service providers, and commercial components.



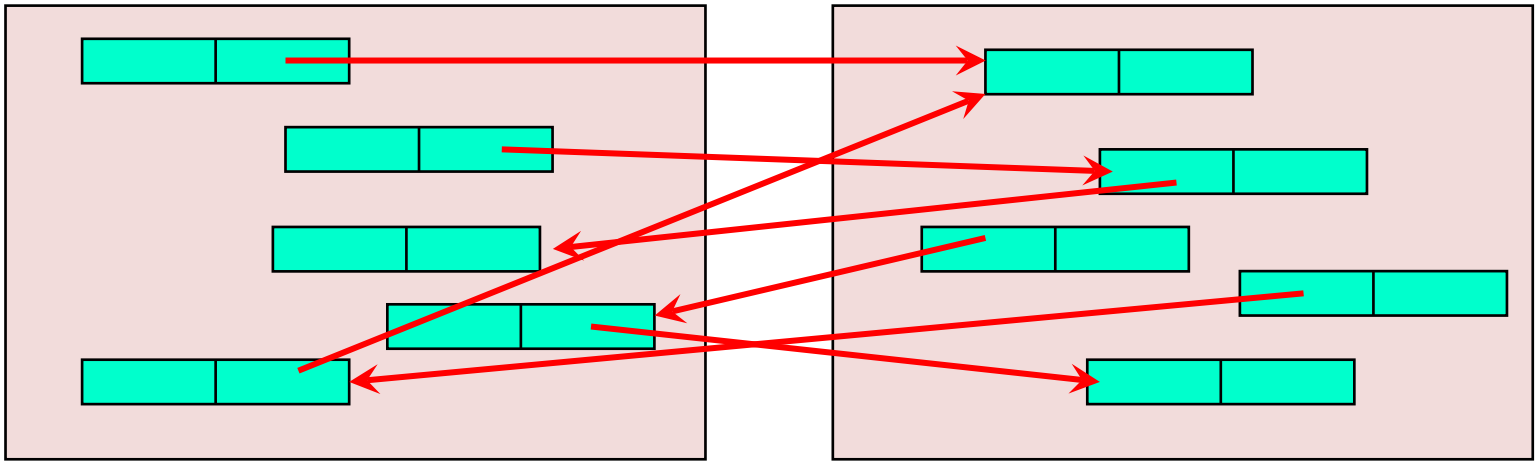
Organization of the KDM metamodel

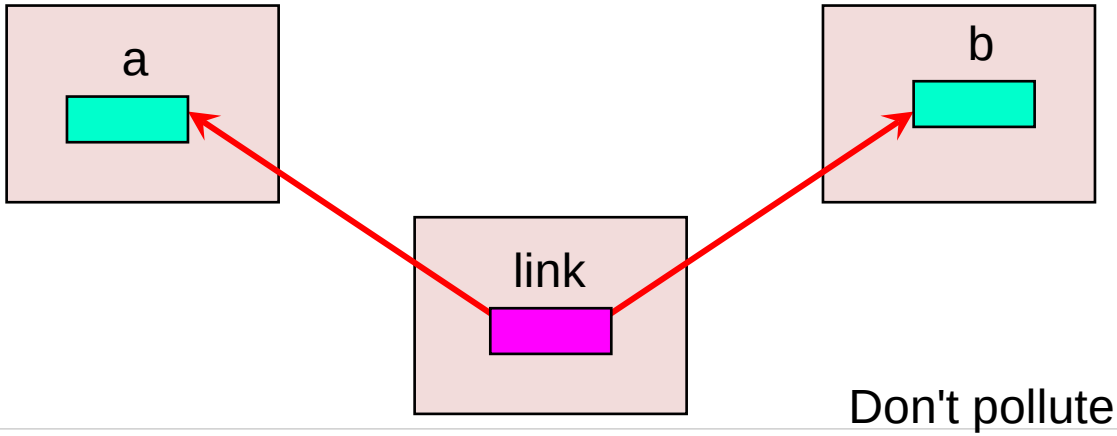
Model Correspondences

- It is often necessary to establish relationships between elements of different models, for several reasons



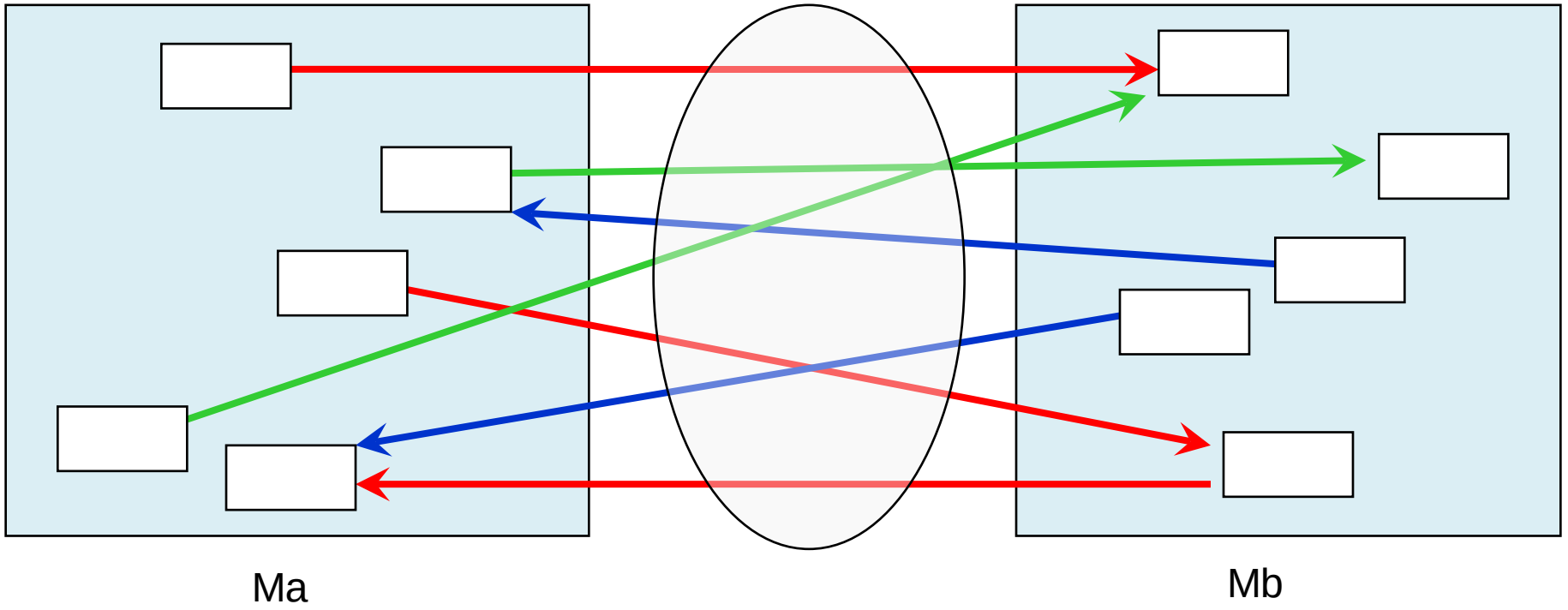
Direct and indirect solutions





Don't pollute my models

An infinity of semantics



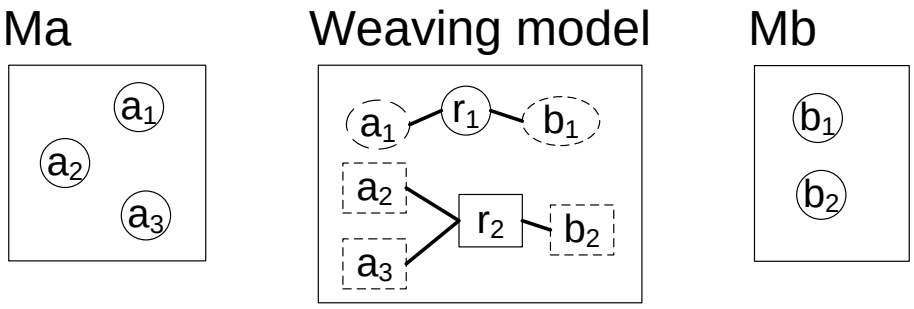
A bunch of connecting lines with different meanings

Three main issues

- What is the form and semantics of the relationships?
  - Cardinality
  - Traceability, merging, equality, annotation, etc.
- How are these relationships created?
  - Manually, automatically, with graphical or textual interfaces
  - In semi-automatic ways (matching heuristic)
  - What heuristics?
  - What cumulative sequence of heuristics?
- How to use these relationships?
  - To trace, to merge, to interoperate, to annotate
  - To follow the dependencies, the provenance

Model weaving (correspondences, mappings)

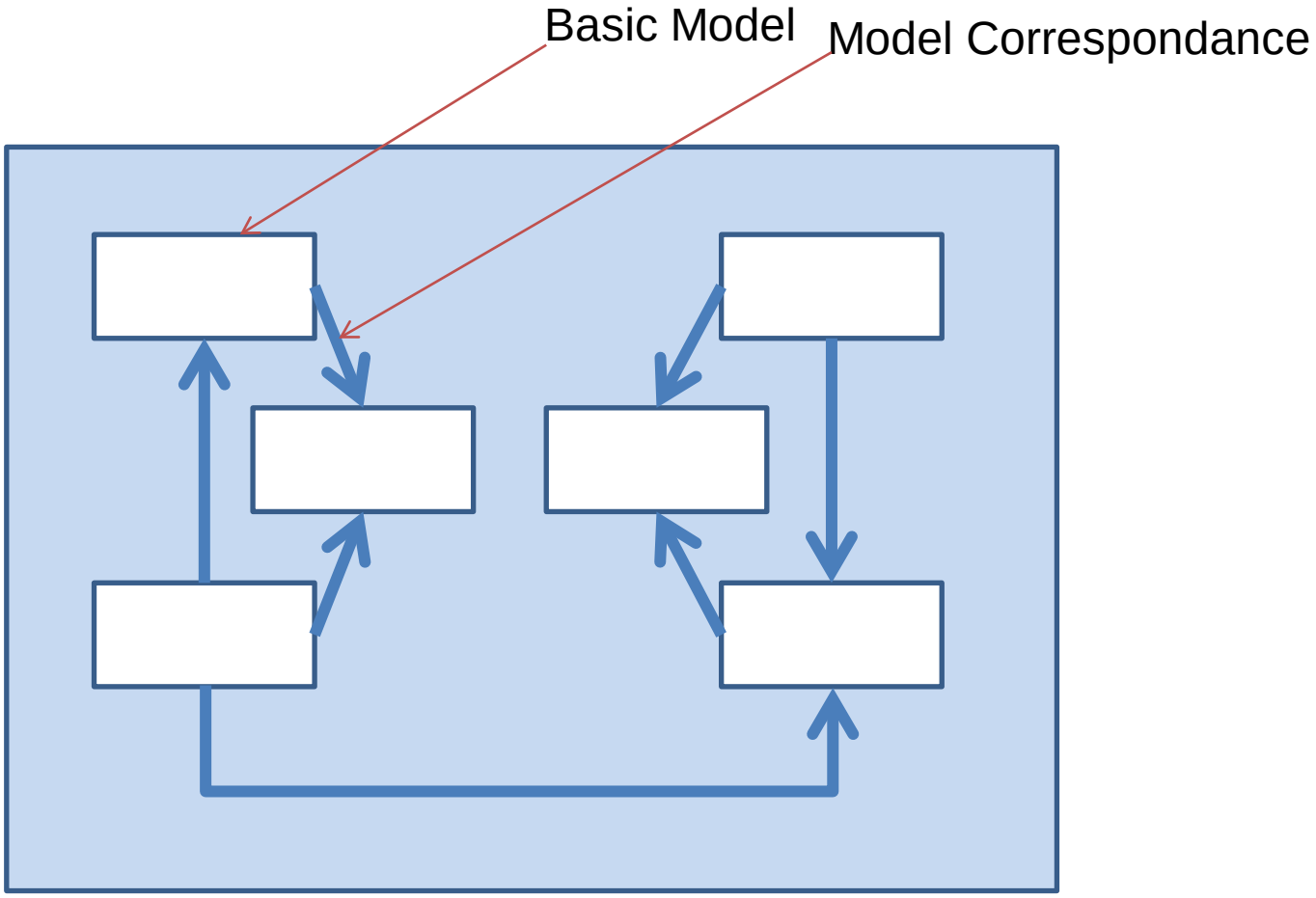
- Related relationships are “grouped" in a weaving model, which is a first class model
  - The model elements represent the relationships and the related elements
  - As any kind of model, the weaving model can be saved, stored, transformed, modified, etc.
- A weaving model conforms to a weaving metamodel
  - Defines the nature of the relationships that can be created
- The weaving metamodel is minimal and extensible
  - Core weaving metamodel (basic link)
  - Extension



Usage of model correspondences

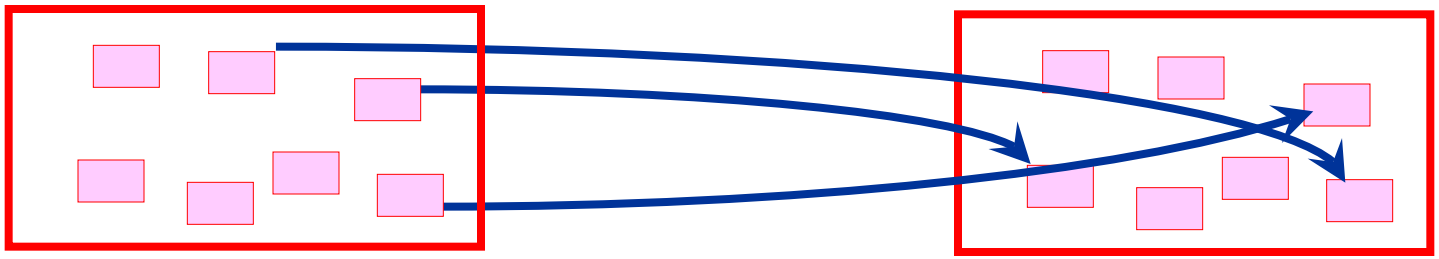
- Traceability/Provenance
  - Weaving models keep track of the source elements used to generate a set of target elements
- Transformation production
  - The model elements are used as specification to produce general purpose transformations
    - ✓ Higher-order transformations translate weaving models into transformation models
- Annotations
  - The weaving model contains extra information used to annotate model elements
- Merging
  - The weaving model is used as input to merge algorithms
- Metamodel comparison
  - The weaving model is used to represent the delta between different models

A model map (megamodel)

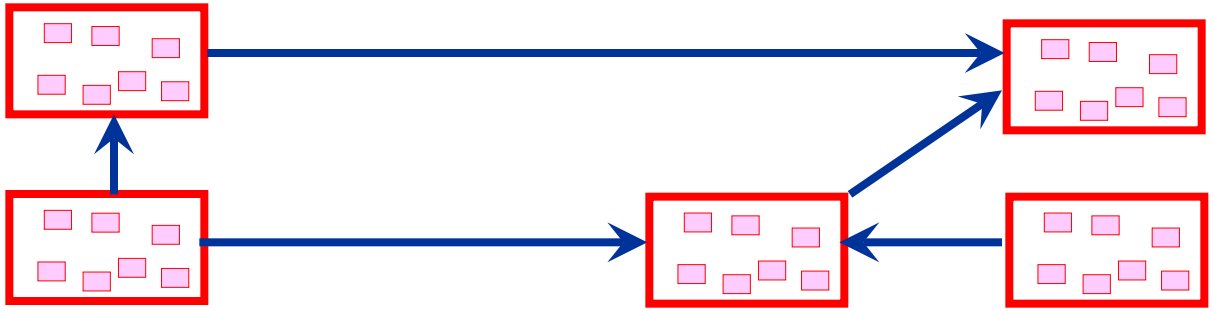


Modeling in the small/large

- Modeling in the small
  - Working at the level of model and metamodel elements



- Modeling in the large
  - Working with models and metamodels as global entities, for what they represent, and their mutual relations, independently of their content
  - A megamodel is a model which elements represents models, metamodels and other global entities (ako model registry with metadata on models and metamodels). A megamodel has a metamodel.



Operations on models

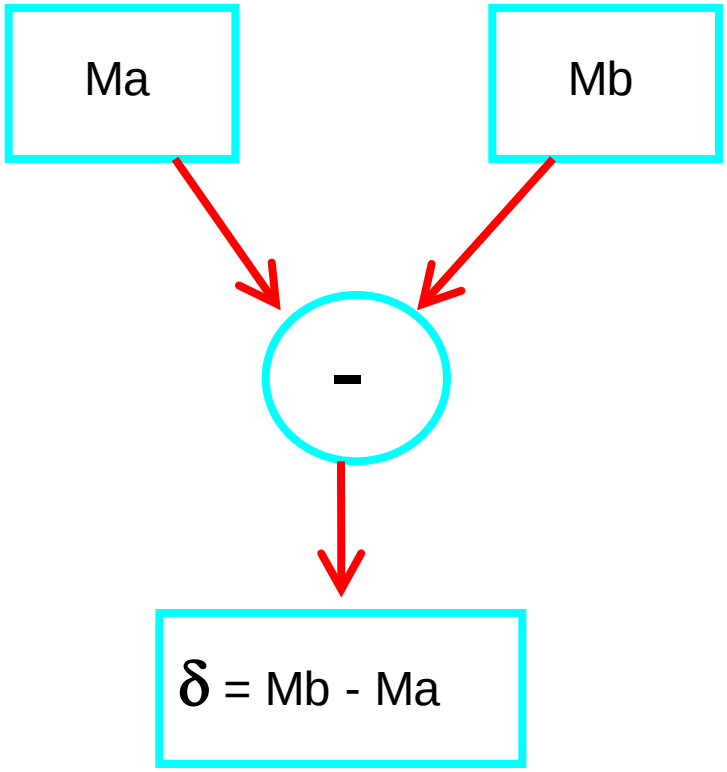
- Unless explicitly stated, apply to all MDE-models
- Most operations are monadic or dyadic operations, often special transformations
  - Create from system or from model
  - Update
  - View (visualize, present visually), Browse
  - Store or Retrieve (in/from a repository)
  - Verify or Measure
  - Merge
  - Align
  - Query or Filter
  - Compare, Normalize
  - Animate or Execute (only executable models)
  - Serialize (total or intermediate)
  - Zoom
  - Slice
  - Transform

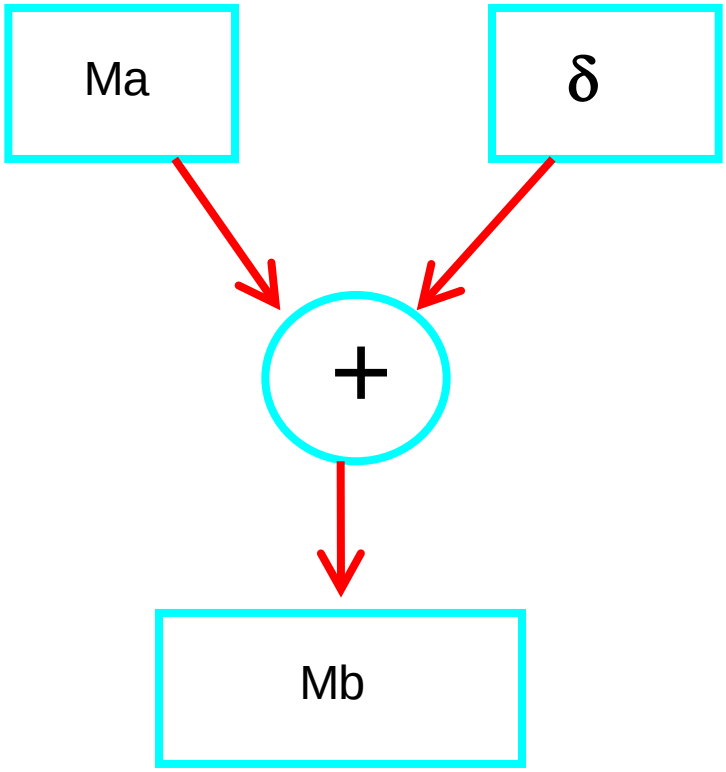
Towards an algebra:

Merge: model × model × mapping → model
Match: model × model → mapping
Diff: model × model → transformation
Split: model → model × model × mapping
Slice: model × criterion → model
etc.

Adapted from "*A Manifesto for Model Merging*" by Steve Easterbrook & al.
http://www.cs.toronto.edu/~gbrunet/pubs/gamma06.pdf

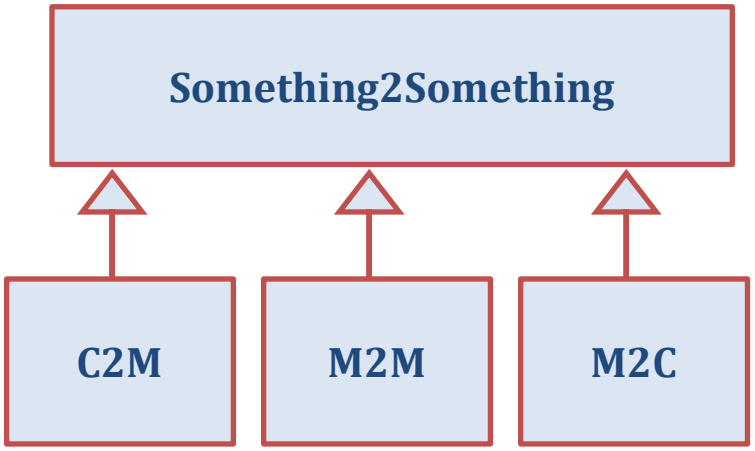
Model diff and compose

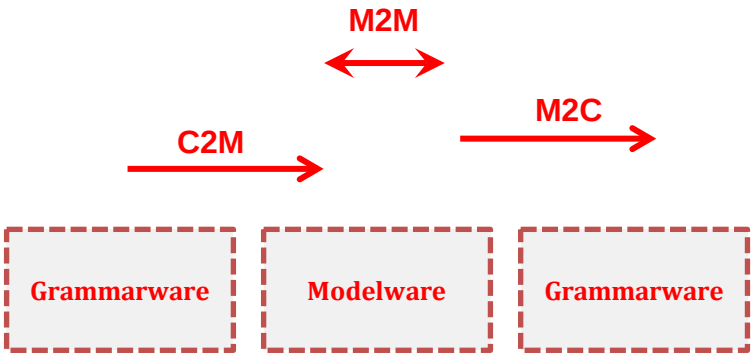




Three main types of transformation

- Three basic situations
  - Model to Model (M2M)
  - Model to System
  - System to Model
- Most common system: code
  - Model to Code (M2C or M2T)
  - Code to Model (C2M or T2M)
- Examples
  - UML -> Javacode
  - Javacode -> Business rules





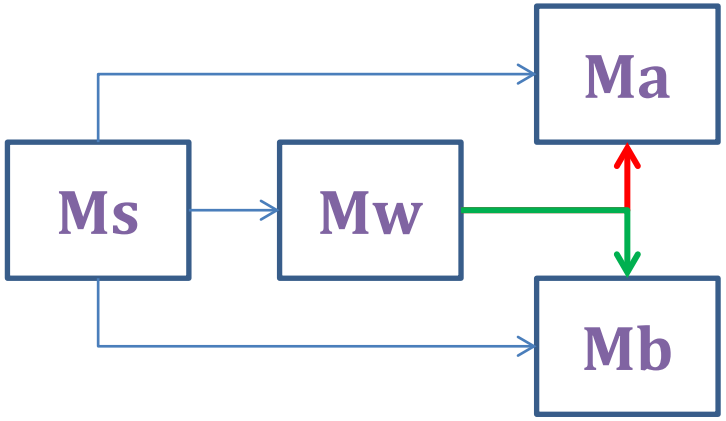
M2M Transformations

- M2M are the internal MDE-TS transformations
- For reference, internal transformations exist in other TSs, for example:
  - XSLT and XQuery for XML-models
  - SQL for RDBMS-models
  - Compilers for Programming-models
  - JSONT for Jason-models
  - SPARQL for RDF-models
  - TMQL for TopicMap-models
- Assuming MDE-Models
  - Taxonomy of Transformation Approaches
  - Taxonomy of Transformation Languages
  - Taxonomy of Transformations

Mb ← f (MMa, MMb, Mt, Ma)

Taxonomy of transformation approaches

- Endogeneous/Exogeneous
- Loss/NoLoss
- Filter
- Separate/Split
- Merge
- Promotion/Demotion (as seen before)
- Unidirectional vs. Bidirectional
- Generation of traceability links (or no)



Bi-directional synchronization (see NII BIG project results)

- Mapping between models established by transformation may be required to be preserved over time =>*synchronization*
- Examples: round-trip engineering, views.
- Propagation of changes to a model may be made in one or more directions.
  - *Strict synchronization*: all changes to models to be taken into account immediately or in the next consistent state, e.g., views.
  - *Loose synchronization:* no statement on when synchronization should occur.
- Bi-directional synchronization is a difficult problem in general.
  - What existing approaches offer solutions?
  - Is bi-directional transformation equivalent to unidirectional transformations in either direction?
  - How can trace information help in the return trip?
  - etc.

http://www.biglab.org/

Taxonomy of Transformation Languages

- **Transformation rules**
  - Defining how elements of a source model should be translated into elements of a target model.
  - A transformation rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS).
  - The LHS accesses the source model, whereas the RHS expands in the target model.
  - They can both be described by variables, patterns, logic, queries, etc. Model transformation approaches differ in how they define rules.
- **Relationship between source and target**
  - Implicitly a new target model is created based on the information in the source model.
  - Approaches exist which do not create new target models but update existing ones or just update the source model.
  - These in-place update approaches can be destructive, e.g. they can remove elements, or they can only made extensions to the existing model.
- **Rule application strategy**
  - There may be more than one match for a rule in a given source scope, => an application strategy is needed.
  - Strategy can be deterministic, non-deterministic or sometimes interactive.
- **Rule scheduling**
  - A scheduling mechanism can be used to determine the order in which individual rules are applied.
  - In some approaches users have no explicit control on the scheduling algorithm.
  - Approaches can also differ in the way rules are selected
- **Rule organization**
  - Modularity mechanisms (packaging rules into modules)
  - Reuse mechanisms (defining rules based on one or more other rules)
  - Organizational structure (organizing rules based on the source or target language)

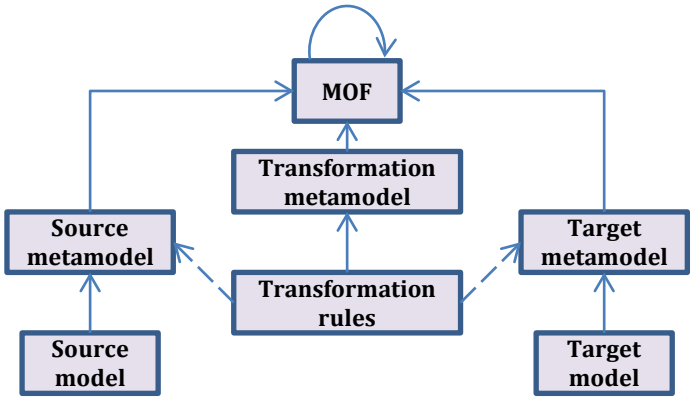
Czarnecki, K., & Helsen, S. (2003). Classification of Model Transformation Approaches. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture. Anaheim, CA, USA.

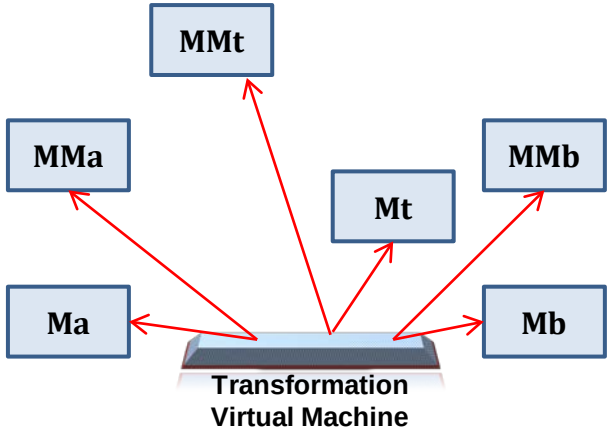
The ATL Execution Engine Architecture

- ATL is specified as a two-part metamodel
  - The structural part defines rules, source and target patterns, imperative blocks, etc.
  - The navigation part is based on OCL 2.0
- A textual concrete syntax is specified using TCS
  - Text ATL can be parsed to its corresponding model
  - An ATL model can be serialized to text
- ATL transformations are models which specify operations on models
  - They can be loaded and serialized by model handlers (EMF, MDR, etc.) or projectors
  - They drive the execution of operations on other models performed using model handlers
- An ATL model can be executed
  - It is first compiled to ATL bytecode: a model-oriented instruction set
  - The ATL Virtual Machine executes this bytecode

Transformations as models

- Treating everything as a model leads not only to conceptual simplicity and regular architecture, but also to implementation efficiency.
- ATL is composed of a transformation virtual machine plus a metamodel-driven compiler.
- The transformation VM allows uniform access to model and metamodel elements.
- Three generations of VMs:
  - Procedure oriented (Wirth's P-machine)
  - Object oriented (Smalltalk bytecode, Java VM)
  - Model oriented (ATL VM, uniform access to models and model elements)





Some ATL transformations

Name	Source Classes	Target Classes	Helpers	Rules	Lines
BibTeXML to Docbook	21	6	4	9	261
Class to Relational	5	4	1	6	245
Java source to Table	5	3	2	2	87
KM3 to DOT	16	26	18	6	624
KM3 to Problem	16	2	6	18	463
PathExp to PetriNet	5	7	20	7	643
Table to Microsoft Excel	3	15	1	3	122
UML to Amble	10 + 10	14	8	11	331
UML to Java	11	8	4	6	80
UML Activity Diagram to MS Project	6	3	3	3	115
UMLDI to SVG	26	38	27	15	2374
XSLT to XQuery	13	18	0	7	165

An example of transformation: KM3 to DOT

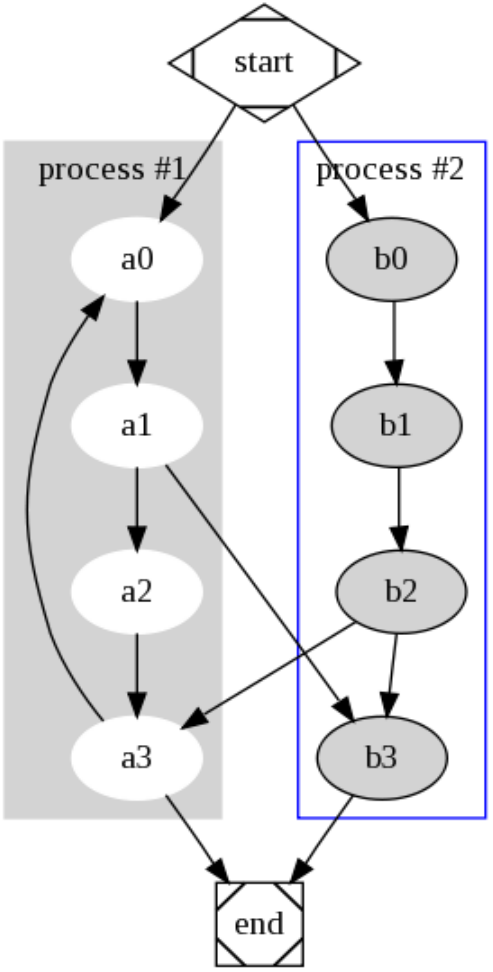
Drawing graphs with *dot*

Emden Gansner and Eleftherios Koutsofios and Stephen North

February 4, 2002

Abstract

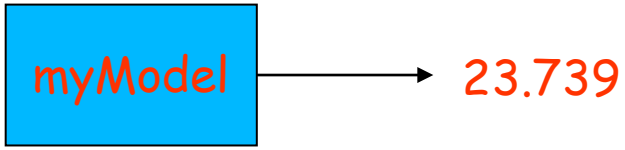
dot draws directed graphs as hierarchies. It runs as a command line program, web visualization service, or with a compatible graphical interface. Its features include well-tuned layout algorithms for placing nodes and edge splines, edge labels, "record" shapes with "ports" for drawing data structures; cluster layouts; and an underlying file language for stream-oriented graph tools. Below is a reduced module dependency graph of an SML-NJ compiler that took 0.98 seconds of user time on a 1.4 Ghz AMD Athlon.

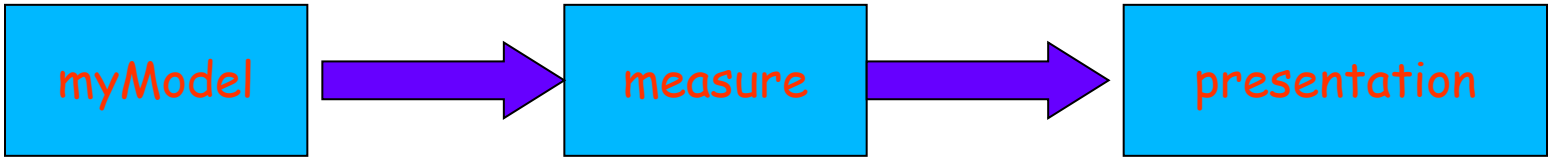


Graphviz is an open source graph visualization software.

Sample of metrics on KM3 Metamodels

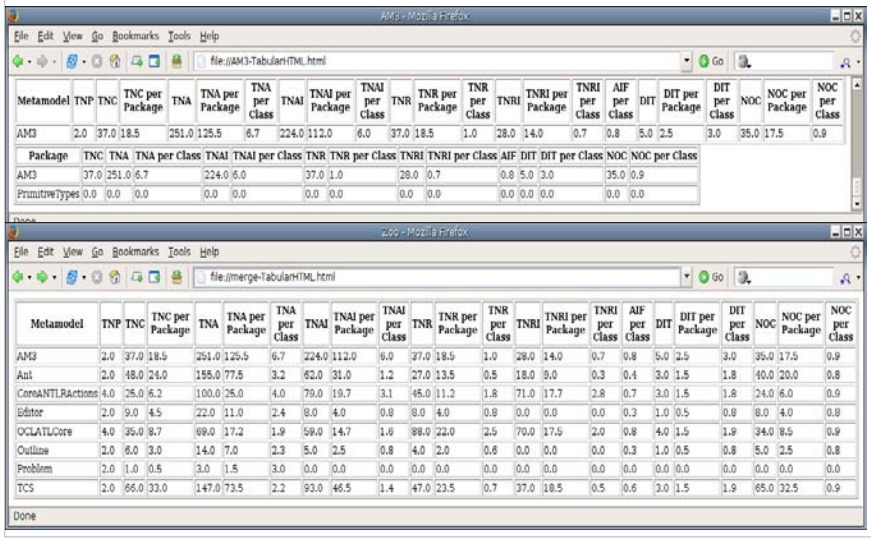
- Total Number of Classes
  - TNC is the total number of classes in a package or a metamodel.
- Depth Inheritance Tree
  - The Depth of inheritance of a class is the DIT metric for a class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.
- Number of Children
  - The Number of Children (NOC) is the number of immediate subclasses subordinated to a class in the class hierarchy.

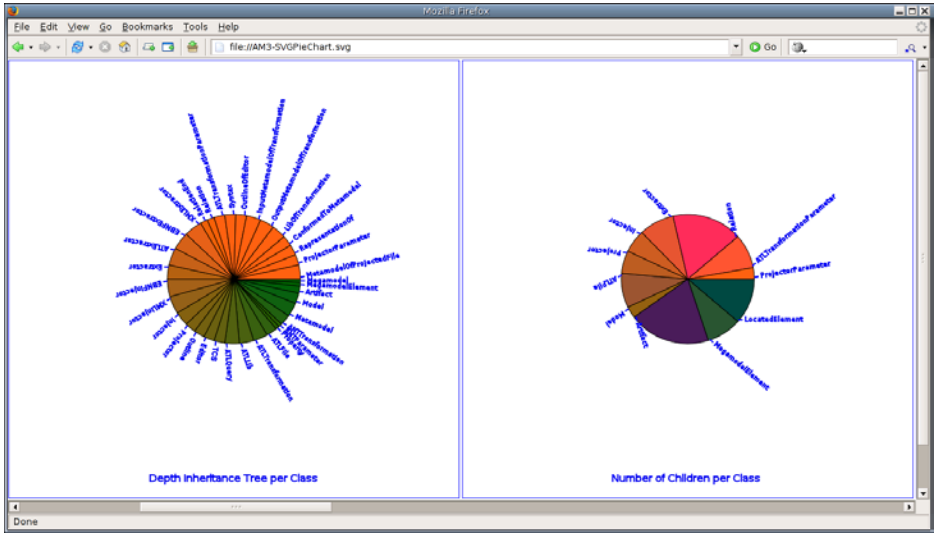




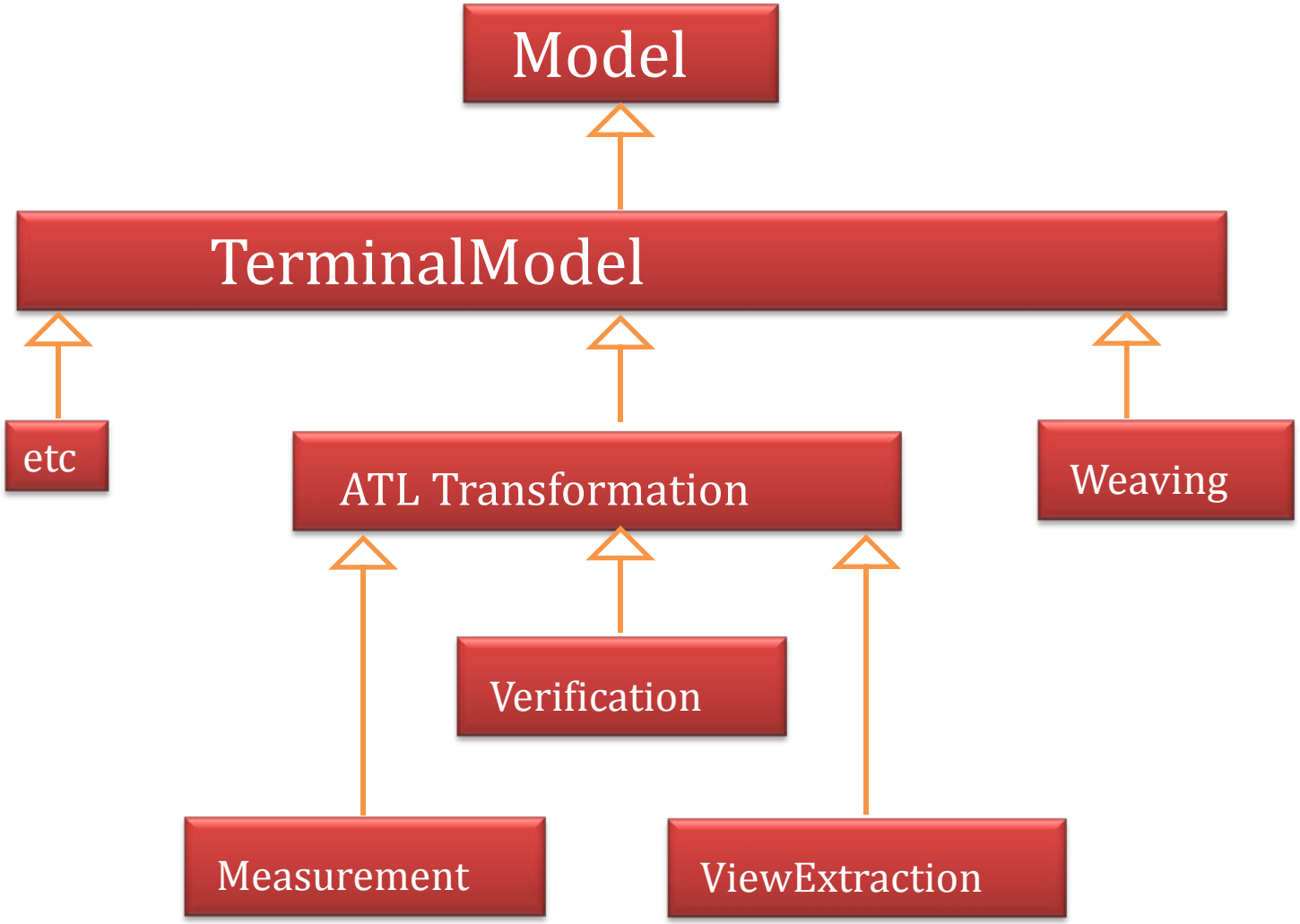
Measures and Presentations

- Different presentation formats for the measurement data collected:
  - Textual representation like HTML tables or Excel file.
  - Graphical representation like SVG charts (Pie, Bar charts).
  - Any other presentation adapted for measurement data.



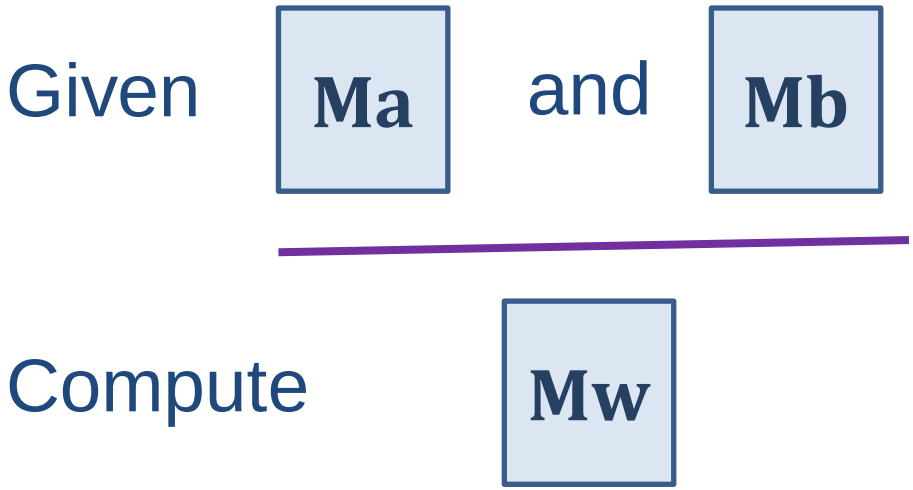


The central role of transformations in MDE

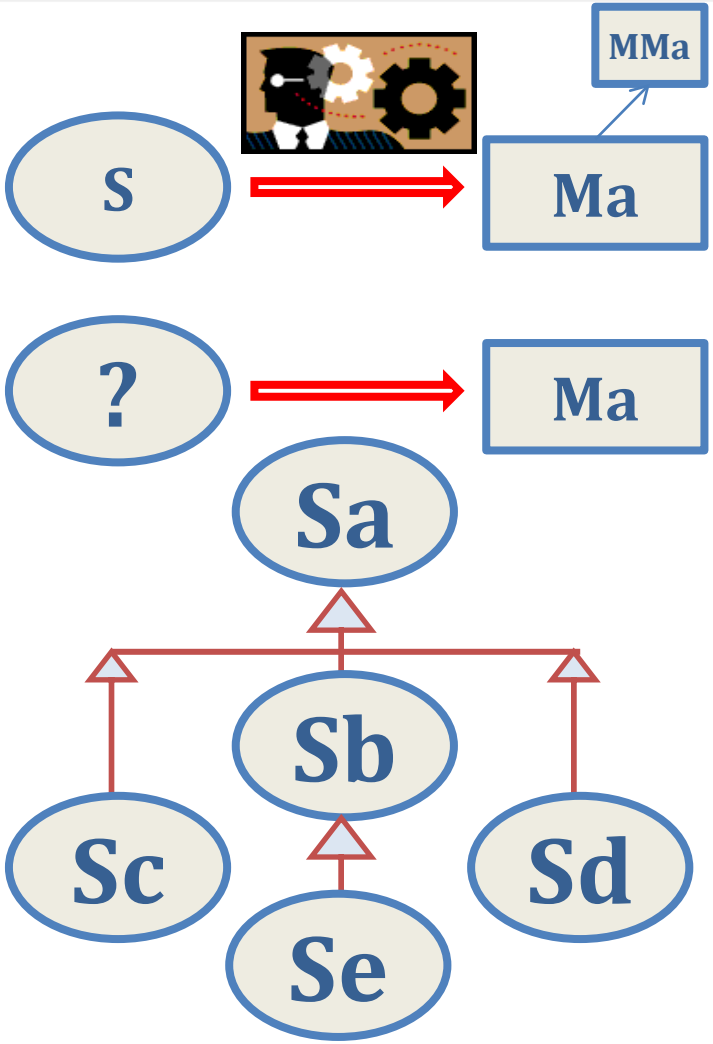


Model Matching

- The automatic or semi-automatic computation of a mapping between two or more models
  - Typical example is the difference betwen to models
  - Many more examples of matching heuristics

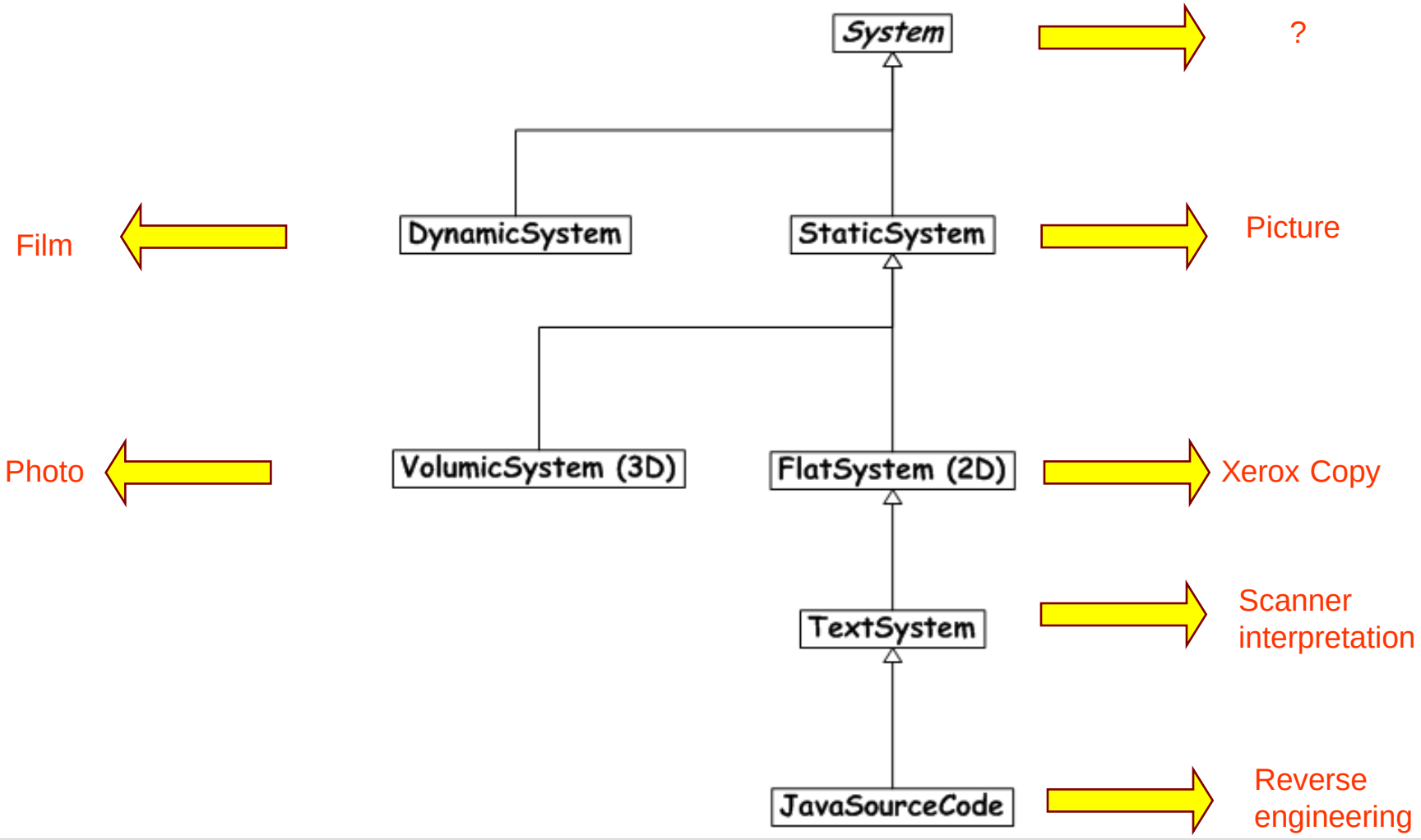


Model Mining

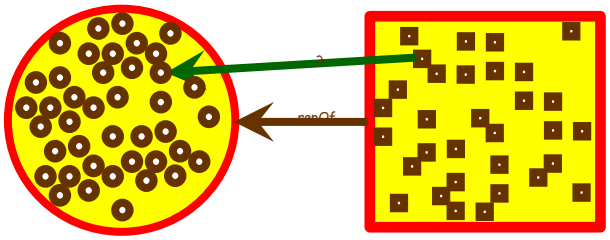


- General model extraction
  - How to derive a model from a system?
- Most difficult situation
  - Nothing known on the system
- The only solution is to consider a classification of systems according to what a priori knowledge we have on them
  - The classification of systems is essential

Model mining depending on the nature of the system



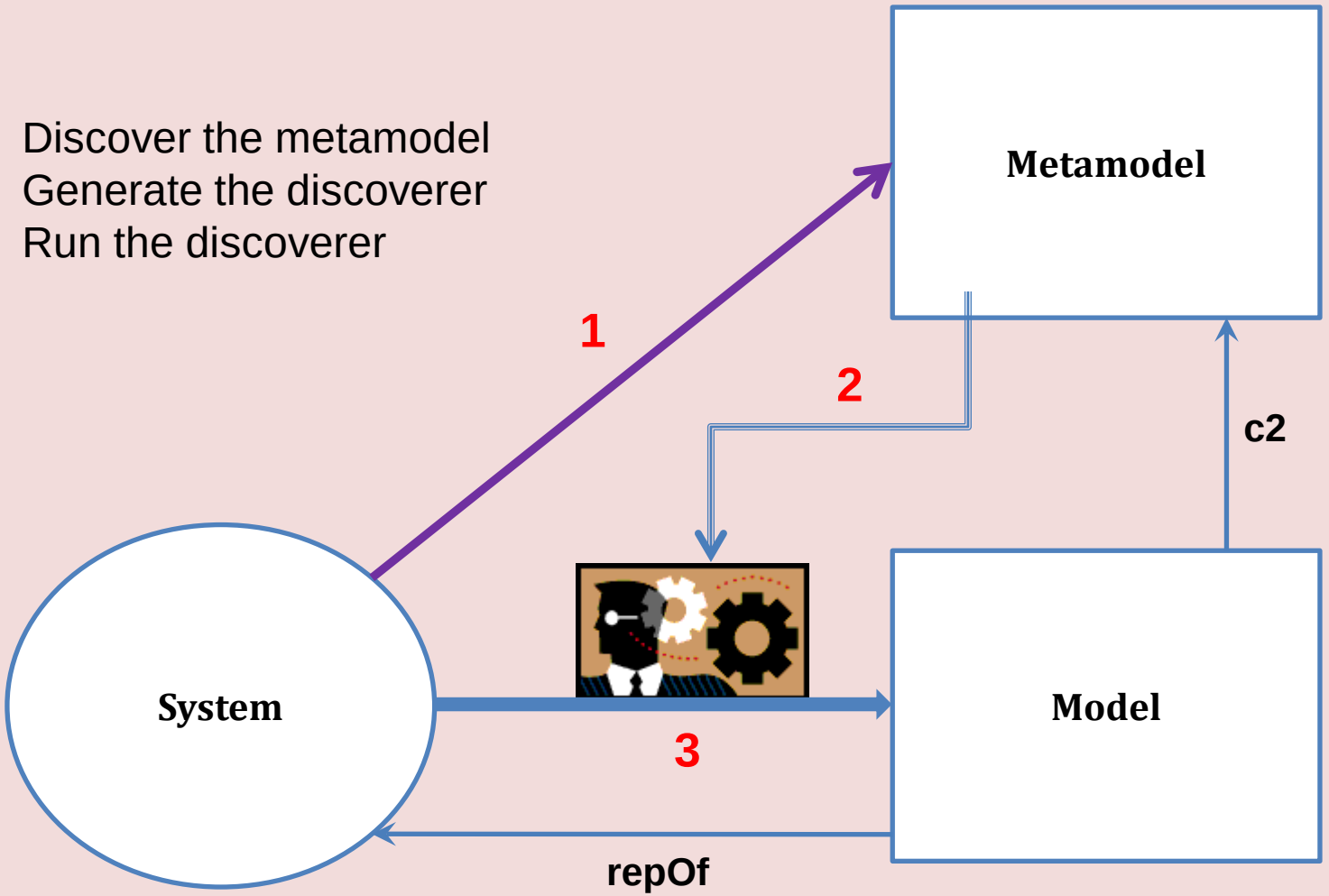
Model Mining methods



- System = {SystemElements}
  - Model= {ModelElements}
- System is code conforming to a given grammar (C2M)
- System is a model in another TS (e.g. XMLware)
- System may have an activable internal observer (e.g. Excel macro)
- System is code in reflective language with introspection (e.g. Smalltalk)
- etc.

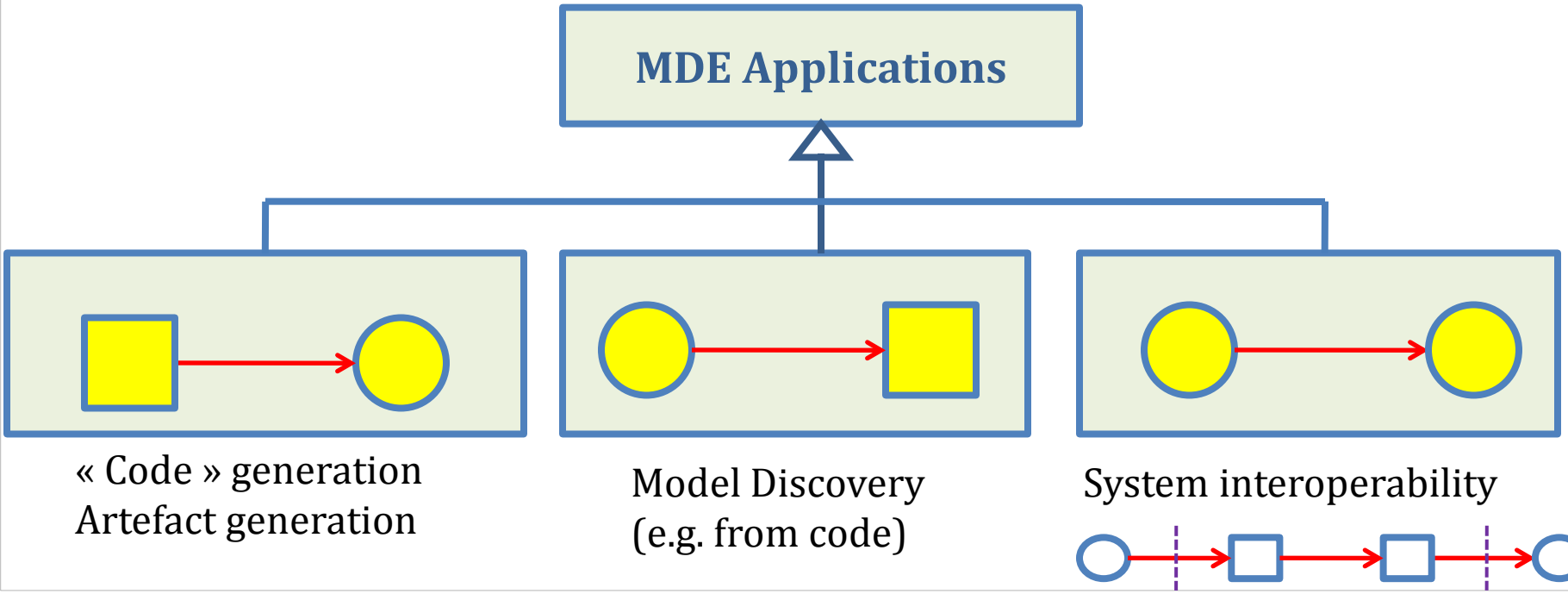
Typical generic discovery process

1. Discover the metamodel
2. Generate the discoverer
3. Run the discoverer



Three main types of MDE applications

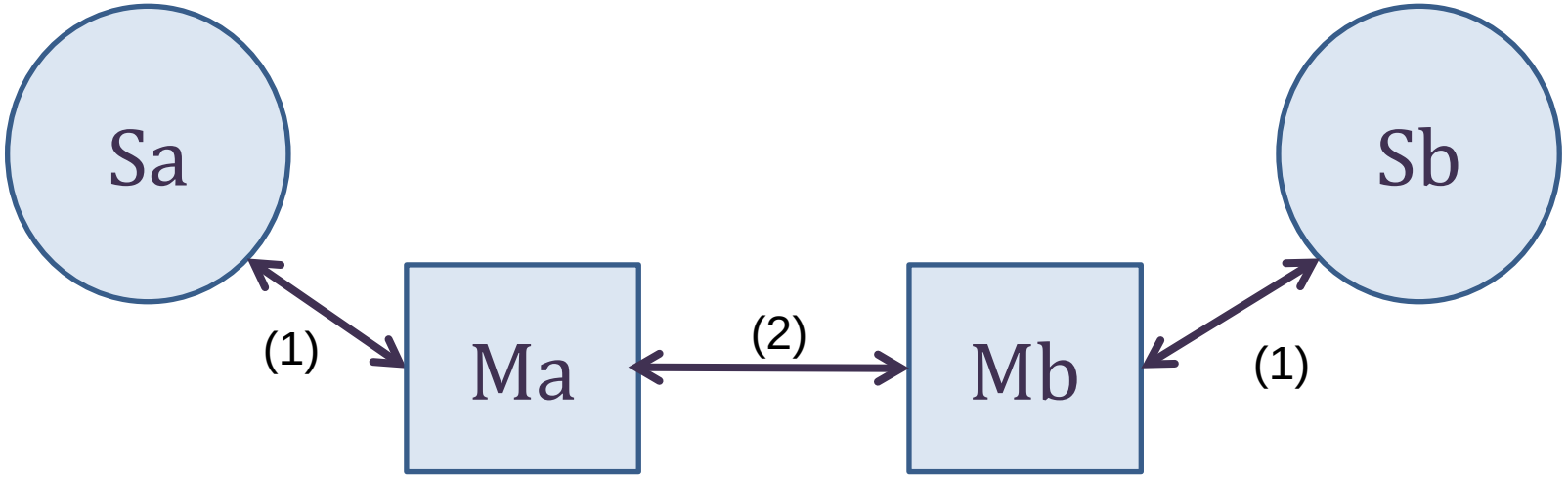
- Three levels of complexity
  - **S ⇐ M** (MD Software Development for development automation)
  - **S ⇒ M** (MD Reverse Engineering for legacy modernization)
  - **S ⇔ M ⇔ M ⇔ S** (Run Time Correspondences for system interoperability)



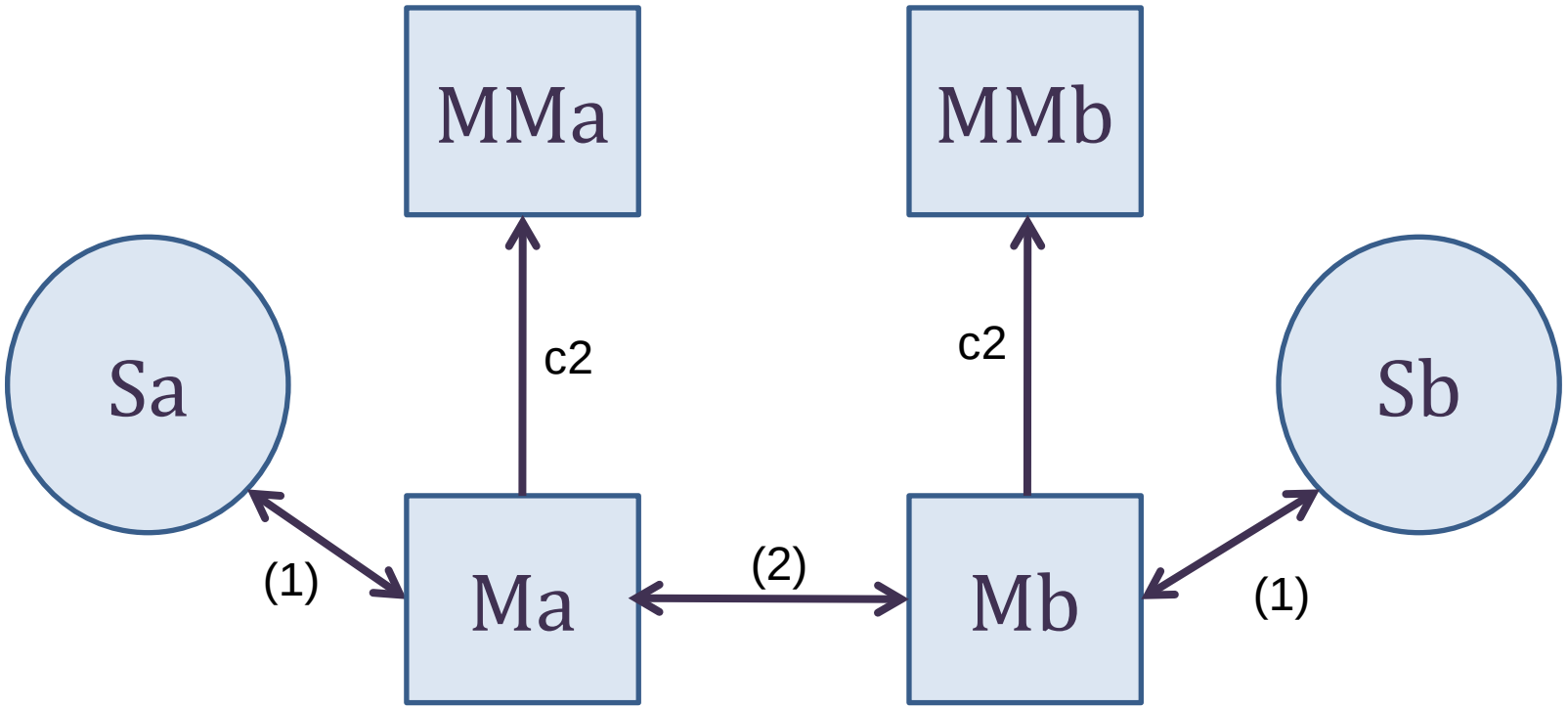
(Model Driven) Interoperability

Two step heterogeneity resolution:

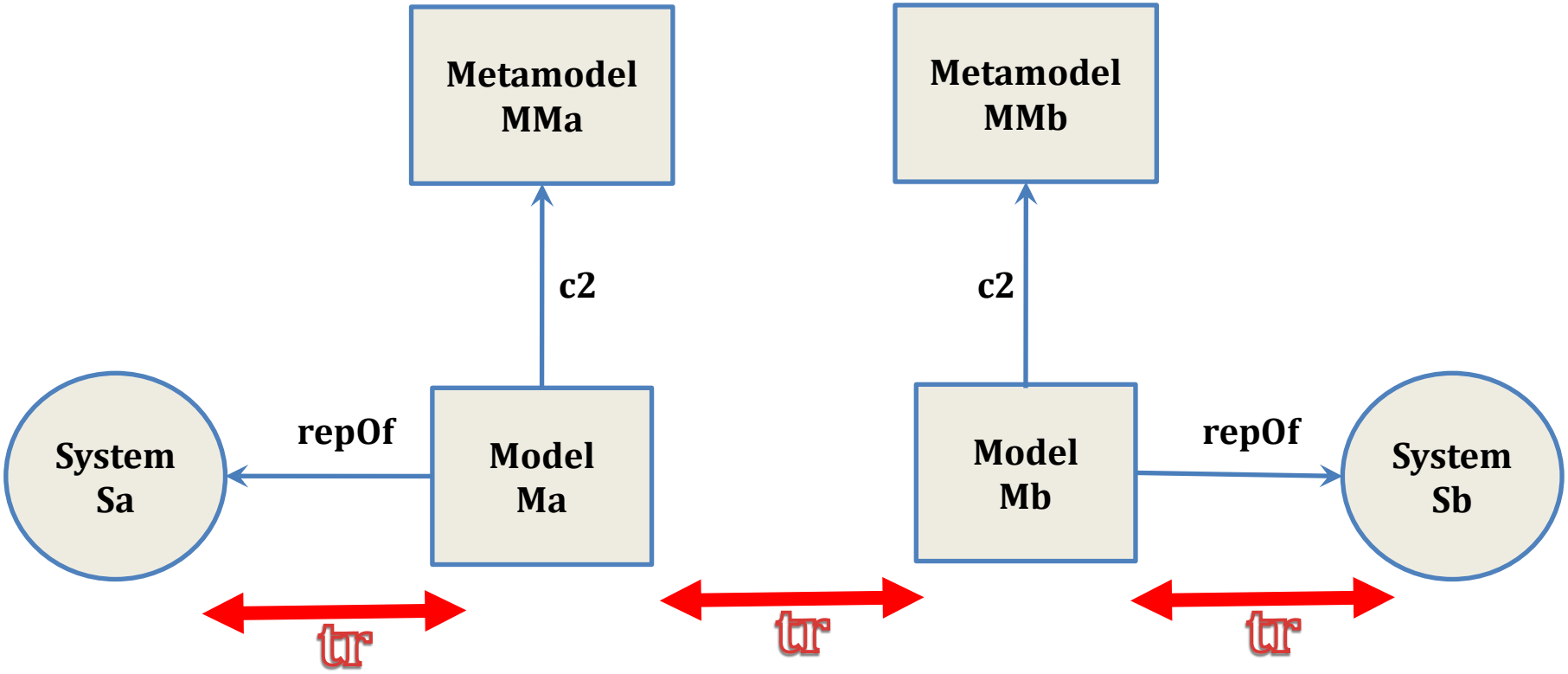
(1) Representing systems by models
(2) Aligning the models



(Language Driven) Interoperability



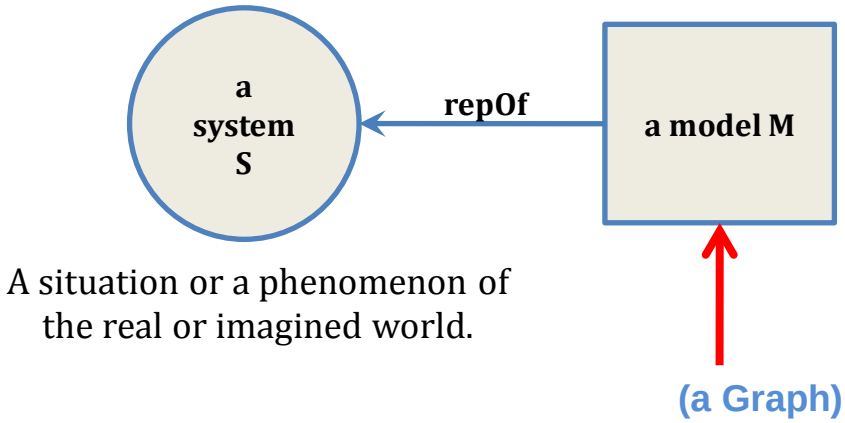
Divide and conquer system interoperability



Two heterogeneous sytems *Sa* and *Sb*
System interoperability implemented by model transformations

Categories of Systems
Types of interoperability

- ❑ Code
- ❑ Data
- ❑ Business
- ❑ Platform
- ❑ Tool
- ❑ Enterprise
- ❑ Requirements
- ❑ Problems
- ❑ Solutions
- ❑ Static systems
- ❑ Dynamic systems
- ❑ Etc.



Need some collaborative cumulative open initiatives.

Example of software tools

- In a given context (company, department, etc.) there are a set of tools used by engineers, for example:
  - Dassault Catia, Microsoft Excel, Microsoft Visio, Rational Rose, OpenOffice, MatLab, Simulink, Bugzilla, Mantis, Doors, Ant, RequisitePro, Reqtify, Protégé, Javacc, Jtest, Flowtracer, Javadoc, Git, Clearcase, Lex, Yacc, Coq, Latex, etc.
- Most of these tools are performing well for their objective, but they have not been built for collaborating mutually , for example:

Taxonomy of Tools, according to MDE

- An ideal MDE tool
  - Strongly associated to one or more metamodels
  - Built with the explicit use of these metamodels (variability of the metamodels is integrated)
  - Change to the metamodel immediatly impact the behavior of the tool
  - An UML modeler should be such a tool
  - The tool consumes and produces models conforming to this metamodel
- Most tools are far from being ideal MDE tools
  - In most situations, the underlying metamodel is implicit and should be made explicit first
  - Then the model import/export should be implemented
  - Then the tool is ready to be MDE-interoperable

Comparison of modeling frameworks

TECHNISCHE UNIVERSITÄT DARMSTADT

Some Related MDD Tools

37

www.es.tu-darmstadt.de

	OMG Languages	AMMA (INRIA)	MS DSL (Microsoft)	GME (Vanderbilt)	Fujaba (UPB,UKa)	MOFLON (TUD)
Metamodel Def. Lang.	MOF	KM3	+	GME 5.0	UML 1.x	MOF 2.x
GUI Def. Lang.	-	-	+	GME 5.0	-	DiaGen
Constraint Def. Lang.	OCL	ATL / OCL	-	-	-	OCL 2.0
Model Trafo Def. Lang.	QVT[1]	ATL	-	GReAT	SDM	SDM
M2M Trans. Def. Lang.	QVT	ATL / AMW[2]	-	GReAT[2]	-	TGG
Code Gen. Def. Lang.	[MOF2 Text]	TCS	+	-	Velocity	Velocity

1: QVT has been designed for model-to-model translation purposes
2: ATL and GReAT are unidirectional model translation languages

ECMDA 2007 - Haifa, June 14, 2007 Real-Time System Lab

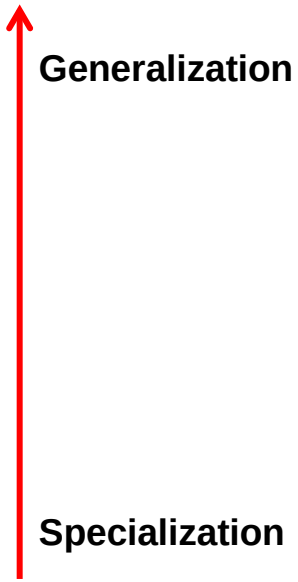
From Andy Schürr guest talk, ECMDA 2007

Scalability

	Terminal Models	Metamodels	Others (e.g. transf.)
Size	<5 000 el. <50 000 el. <500 000 el.	<50 el. <5 000 el. <50 000 el.	<500 rules <5 000 rules
Evolutivity	Every second Every minute Every hour Every day Every year Never	Every second Every minute Every hour Every day Every year Never	Every second Every minute Every hour Every day Every year Never
Heterogeneity	XMI, native data, etc.	MOF, DSL, KM3, etc.	QVT, Viatra, ATL, XSLT, etc.
Quantity	1 10 100 1 000 10 000	1 10 100 1 000 10 000	1 10 100 1 000 10 000

Main message of the presentation

- MDE has evolved a lot since its inception in 2000
- In order to analyze rationally this going-on evolution, we need to look carefully at the classification of MDE artifacts
- From information systems to embedded systems, from Web engineering to open data engineering, the same basic kind of operations are used
- Powerful operations are available at the top of the generalization/specialization hierarchy axis (store, retrieve, transform)
- More specific operations may be defined at the bottom of this axis



Thanks

- Questions?
- Comments?

JBezivin@gmail.com