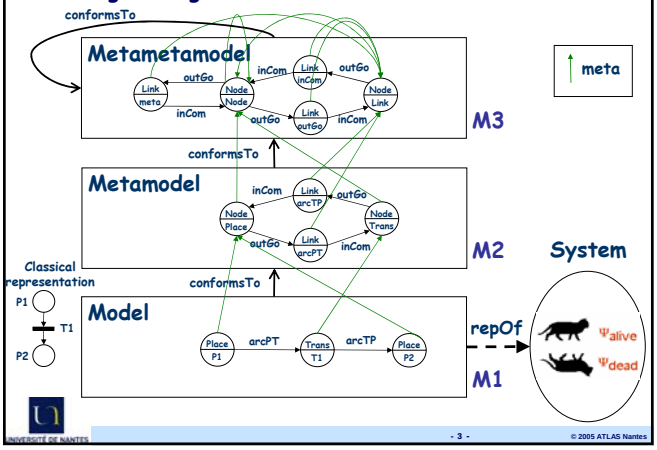
Model Transformation and Weaving Tools in the AMMA Platform

Jean Bézivin, Marcos Didonet Del Fabro, Frédéric Jouault
{jean.bezivin, marcos.didonet-del-fabro, frederic.jouault}@univ-nantes.fr

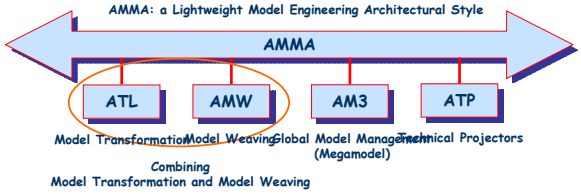
ATLAS group (INRIA & LINA), University of Nantes, France
http://www.sciences.univ-nantes.fr/lina/atl/

Contents

Model Engineering Overview



AMMA Platform Overview

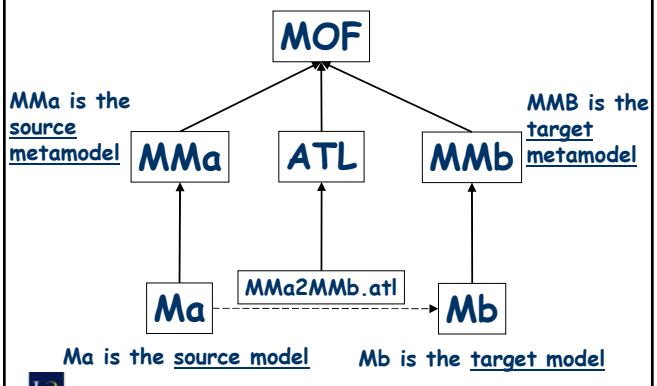


Contents

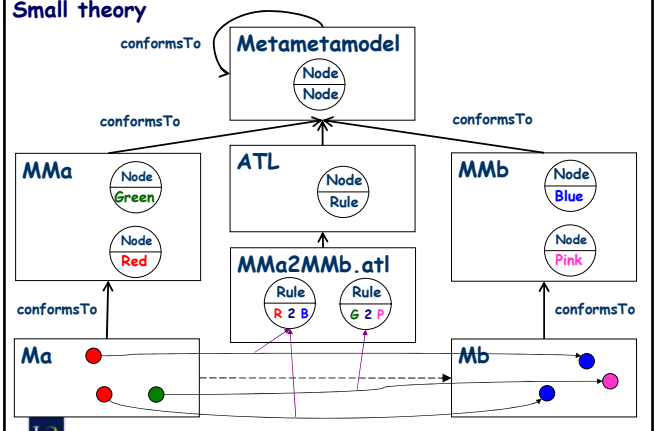
Model transformation definition

- By definition, a model transformation is the automatic creation of target models from source models.
- Model transformation is not only about M1 to M1 transformations:
  - M1 to M2: promotion,
  - M2 to M1: demotion,
  - M3 to M1, M3 to M2, etc.

Some definitions



Small theory



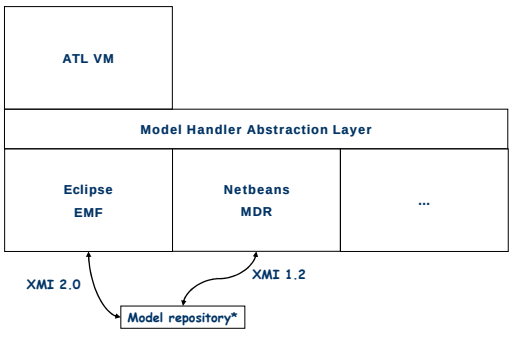
ATL Overview

- Source models and target models are distinct.
- Source models cannot be modified, only navigated.
- Target models cannot be navigated.
- The language is a declarative-imperative hybrid:
  - There are declarative matched rules.
  - There are imperative called rules (to be implemented).
- An imperative rule is basically a procedure.
- A declarative rule specifies:
  - a source pattern to be matched in the source models,
  - a target pattern to be created in the target models for each match.

Execution order

- The order in which rules are matched and applied is not specified.
- The order in which bindings are applied is not specified.
- The execution of declarative rules can however be kept deterministic:
  - The execution of a rule cannot change source models: it cannot change a match,
  - Target elements are not navigable: the execution of a binding cannot change the value of another.

The ATL Virtual Machine Architecture

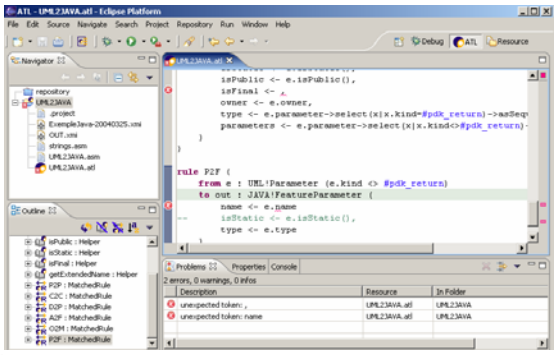


*The simplest form of a model repository is a file system.

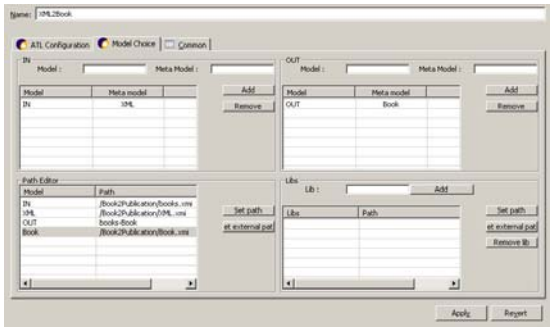
ATL Development Tools (ADT): an Eclipse-based IDE for ATL

- ATL perspective and ATL project
- ATL Code editor
  - Syntax highlighting
  - Outline
  - Error reporting
- ATL Builder
- ATL Debugger
  - Source-level debugging
  - Stepping through elementary operations
  - Breakpoints support

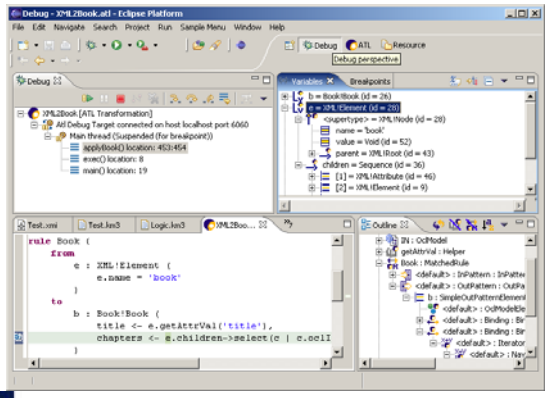
ATL Development Tools: perspective, editor and outline



ATL Development Tools: launch configuration



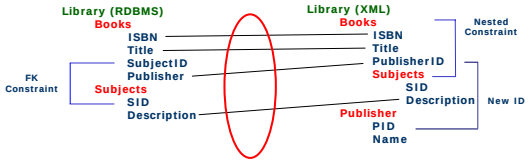
ATL Development Tools: source-level debugger



Contents

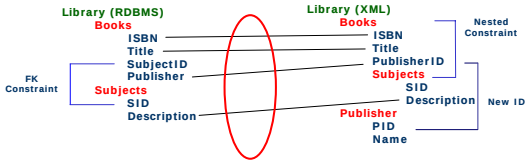
Model Weaving

- Model Weaving means:
  - Establishing typed links between model elements
  - Representing these links as models
  - Defining link types in a metamodel



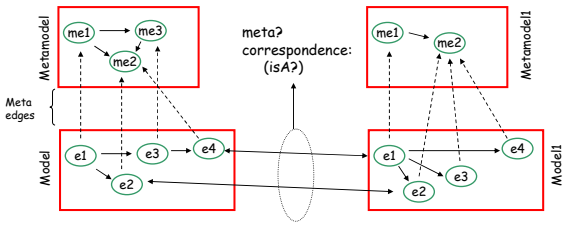
Model Weaving Use Cases

- Data exchange
- Schema evolution
- Heterogeneous data integration
- Mapping between models and a graphical representation
- Transformation specifications
- Etc.



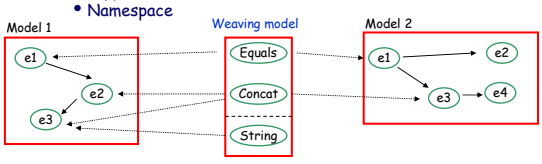
Models

- Models: directed graphs
- Mappings: how to represent them?



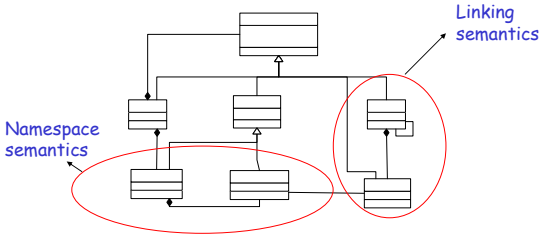
Weaving model

- Correspondences as first class entities and with semantics
  - Conforms to a weaving metamodel
  - Weaving link
    - concatenation, ordered, equals, containment, supplier/consumer, etc.
    - Namespace



Weaving core metamodel

- Several weaving metamodels (extending a minimal metamodel)
  - we cannot predict all weaving semantics
    not a general purpose weaving language



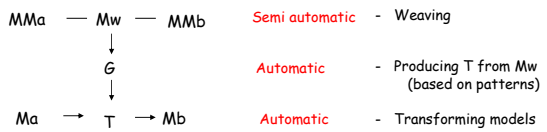
Weaving as models: advantages

- Use of same model management primitives
  - Add, delete, update, navigation
- Expressiveness
  - Reason about mappings (infer about mappings)
    ("The fundamental role of a reasoning engine is to derive new information via automated inference"[sigmod04])
  - Foreign keys, nested constraints, ordering, concatenation, merge, etc.
    - Hard to express with 1-1 correspondences
      (semantics implemented ad-hoc)

Weaving as models: advantages

- Adaptability
  - Minimal metamodel + extensions
    - Namespace extensions, matching based extensions, concatenation, ordered, equals, containment, supplier/consumer
- Efficiency and reuse
  - Weaving model is a specification, it is not executable
    - Same weaving model used to produce:
      - ATL -> model transformation language
      - XSLT -> XML
      - Morphisms -> generic model management
      - SQL -> relational databases

Efficiency and reuse: weaving model + transformations



model

WModel

WRef
-ref : String

WLin

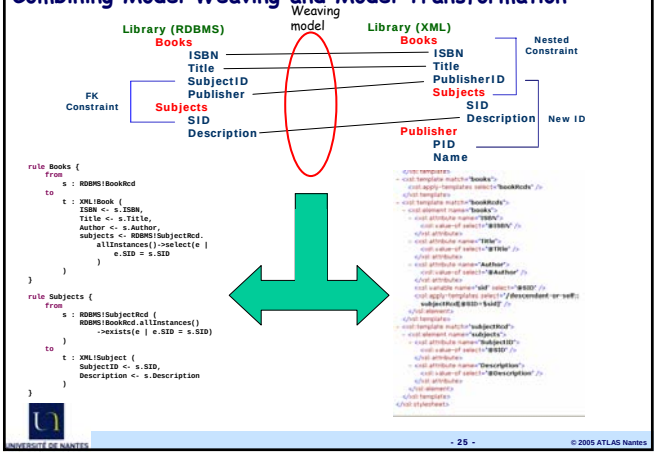
link

wovenModel (1-*)

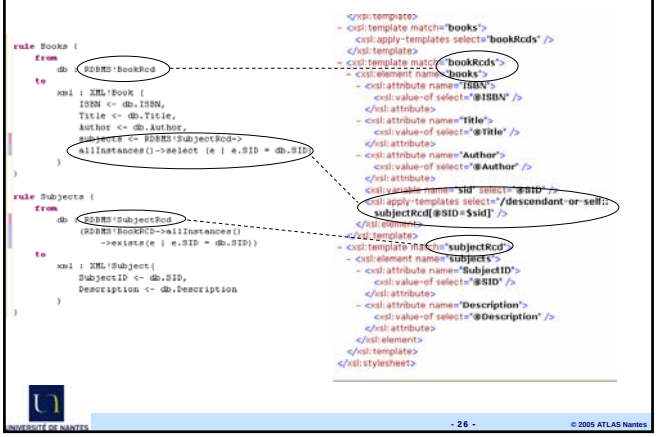
WModelRef

WElementRef

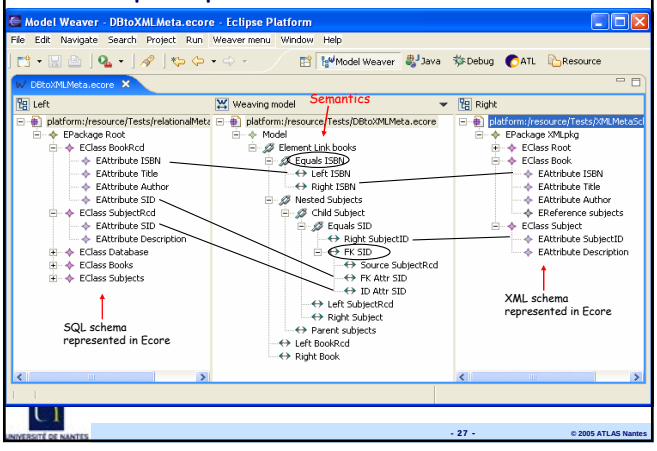
Combining Model Weaving and Model Transformation



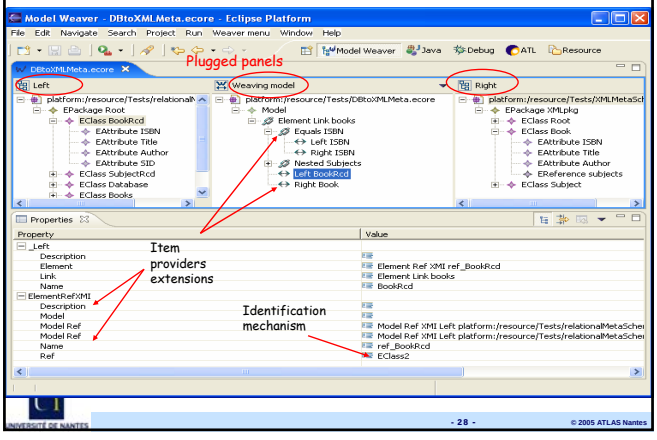
For a given weaving model, two representations: ATL and XSLT



AMW Example: represented semantics



AMW Example: some UI facilities



Contents

- Introduction
- Model Transformation with ATL
- Model Weaving with AMW
- Case Study: Bridging UML Profiles and Domain Specific Languages
  - Overview
  - UML Profiles
  - Implementation Architecture
- Demonstrations

Bridging UML profiles and Domain Specific Languages*

- Two major approaches appear in Model Engineering to model domain specific concepts:
  - UML approach : uses a single, large, general purpose metamodel, extensible through profiles.
  - DSL (Domain Specific Languages) approach: uses a large number of small, domain specific, metamodels.
- We need interoperability between these approaches:
  - A tool which can perform automatic transformations between profiled UML models and models conforming to corresponding metamodels.

*Work realized by Anas Abouzahra.

UML Profiles (1/2)

- UML profiles are dedicated to the strategic intention to formalize and support the development of applications with UML:
  - It allows designers to customize UML to their particular domain or purpose .
  - It defines virtual UML subclasses by associating **stereotypes**, **tag definitions** and **constraints** to provide an additional meaning to UML classes.
- Profiles are generally represented by mapping tables which show correspondences between stereotypes and concepts.

Mapping table

MetaModel Element	UML Element
MetaModel_Class	The UML class and optional stereotypes, which combination represent the MetaModel_Class

UML Profiles (2/2)

- Each element of the profile maps to a specific metamodel element
- The mapping table is completed by mapping details:
  - They define more precisely each map between UML and metamodel elements.
  - Each mapping details contains subsections covering these topics: tags, mapping properties, constraints, and limitations.
  - Tags are used for metamodel properties not directly supported by UML.
  - Some domain specific details cannot be rendered in UML using profiles. Those details are described in a *limitations* section.

Tags

Tag	Value

Feature Map

MOF Feature	UML

UML Model Constraints

Limitations

Example: UML Profile For MOF (1/2)

- Purpose: using widely available UML CASE tools to design metamodels
- Each element of the profile maps to a specific MOF element.

UML Profile For MOF: concepts mapping table

MetaModel Element	UML Element
Package	Model or Package, both with <<metamodel>> stereotype
Association	Association
Exception	Exception or Class with <<exception>> stereotype
Constant	DataValue
Constraint	Constraint
Import	Dependency with <<import>> or <<clustering>> stereotype
PrimitiveType	DataType
Reference	Attribute with <<reference>> stereotype or AssociationEnd (if implicitReferences is set to true)

Example: UML Profile For MOF (2/2)

- Each element of the profile maps to a specific MOF element as shown in the mapping table above.
- The mapping table is completed by mapping details.

Tags

Tag	Value
org.omg.uml2mof.hasImplicitReferences	true (default) or false

Feature Map

MOF Feature	UML
container	namespace or null if the namespace is either null or not mapped to a MOF Package
contents	ownedElement, taggedValue
isRoot	isRoot
isLeaf	isLeaf
isAbstract	isAbstract
visibility	always set to public_vis
supertypes	other packages on supplier end of UML dependencies stereotyped as <<subtyping>> that binds them to this package.

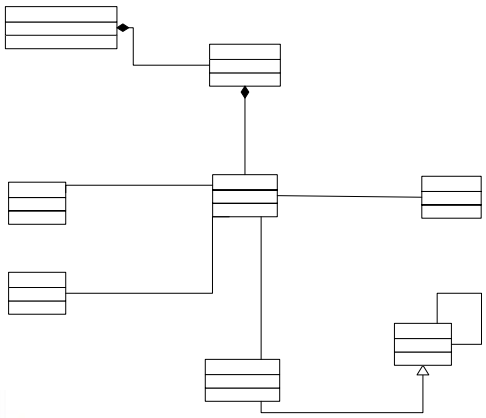
UML Model Constraints

UML Model/Package representing a nested MOF Package must not have a tag org.omg.uml2mof.hasImplicitReferences.

Limitations

The order of elements is not fully preserved when rendered using the profile as UML since it has separate associations for ownedElement and taggedValue.

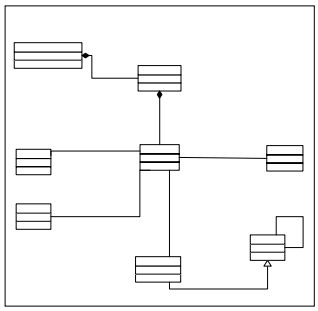
Profile metamodel



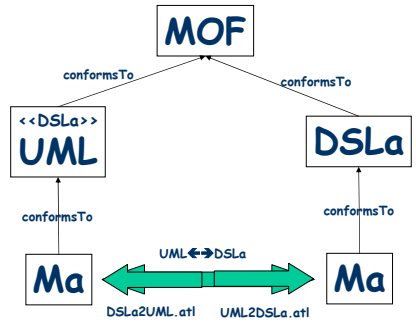
Profile metamodel

- The definition of each virtual UML subclass which corresponds to a definition of a Profile element (in particular a class of the metamodel profiled) can be done in several ways.
- An UML element can take more than one stereotype to define the same element of the metamodel profiled. However, it can also take different stereotypes to define different elements.
- We should have the possibility to navigate the UML elements used by the Profile and have access to some of their attributes and values

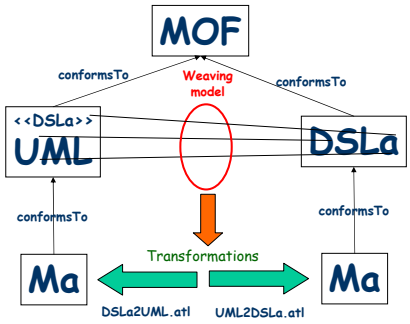
Profile MetaModel



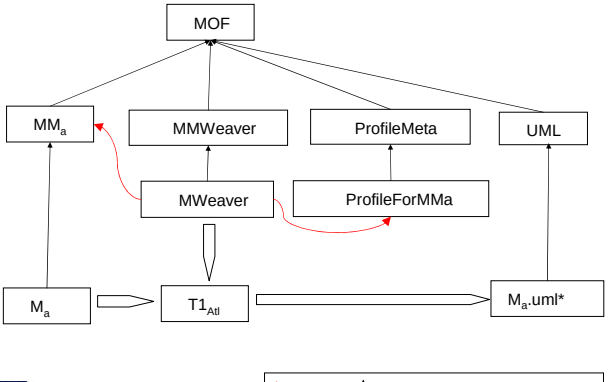
Implementation Architecture



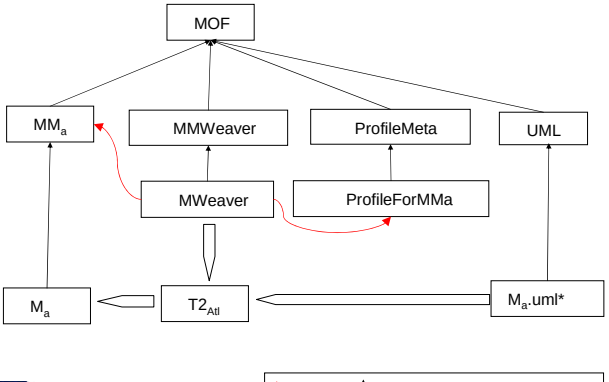
Implementation Architecture



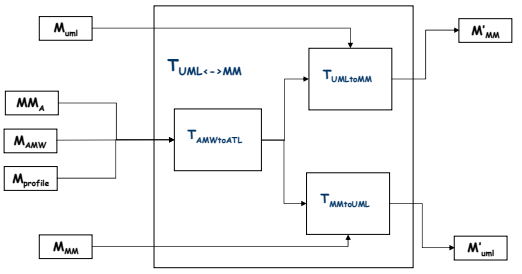
Using ATL transformations (1/3)



Using ATL transformations (2/3)



Using ATL transformation (3/3)



$$\{M'_{uml},\ M'_{MM}\} = T_{UML<->MM}(MM_A, M_{AMW}, M_{profile}, M_{MM}, M_{uml})$$

$$M'_{uml} = T_{MMtoUML}(M_{MM})$$

$$M'_{MM} = T_{UMLtoMM}(M_{uml})$$

$$\{T_{UMLtoMM},\ T_{MMtoUML}\} = T_{AMWtoATL}(MM_A, M_{AMW}, M_{profile})$$

The ProfileForMOF Model

- profile ProfileForMOF : MOF {

.....

define AssociationEnd {
UML!AssociationEnd {
tag documentation ;
-- AssociationEnd.annotation
}
nested{
ref name;
ref namespace;
ref constraint;
ref participant;
ref multiplicity {range { lower , upper } };
ref ordering : literal {ok_ordered (true) , ok_unordered(false)} ;

-- isOrdered is mapped to ordering (where true corresponds to ordered)

ref isUnique : boolean = true;
ref aggregation : literal {ak_composite , ak_aggregate ,ak_none};
ref isNavigable;
ref changeability : literal {ck_changeagle (true), ck_frozen(false) , ck_addOnly(false)};
}
}

.....

}

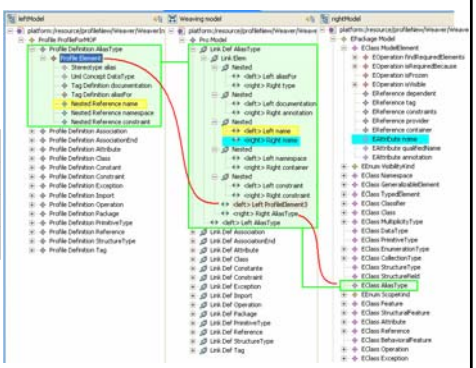
Tag	Value
documentation	any string

MOF Feature	UML
name	ModelElement.name
annotation	body of the first comment in ModelElement.comment; value of documentation tag
container	ModelElement.namespace
constraints	ModelElement.constraint

Weaving model of the profile for MOF

Tag	Value
documentation	Any string
alyasFor	true or false

MOF Feature	UML
name	ModelElement.name
container	ModelElement.namespace
constraints	ModelElement.constraint



Contents

End of the presentation

- Thanks
  - Questions?
  - Comments?

AMMA@lina.univ-nantes.fr
ATLAS group, INRIA & LINA, Nantes