



Meta-model Pruning

Sagar Sen, Naouel Moha, Benoit Baudry, Jean-Marc Jézéquel

Equipe TRISKELL, INRIA Rennes Bretagne-Atlantique

October 3-9, MODELS 2009

Denver Colorado

Outline

- **Where do large meta-models come from?**
- Why a need to extract subsets of large meta-models ?
- How to prune a meta-model to obtain subsets called effective meta-models ?
- What are the type properties of effective meta-models ?
- Illustrative Case Study : French Space Agency
Transformation Input Meta-model
- Conclusion



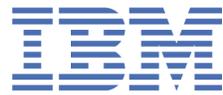
Where do large meta-models come from?

A widely used large meta-model is that of a
General Purpose Modelling Language such as the **UML**



Why is the UML meta-model large? (1)

Member Space



About **389 official** participating stakeholders...

Contribute to / use...



Why is the UML meta-model large? (2)

Tool Space

Editors:

Topcased,
StarUML,
ArgoUML...

Code generators:

Acceleo,
BOUML,
Eclipse..

Transformation Tech:

Kermeta
EMF
ATL

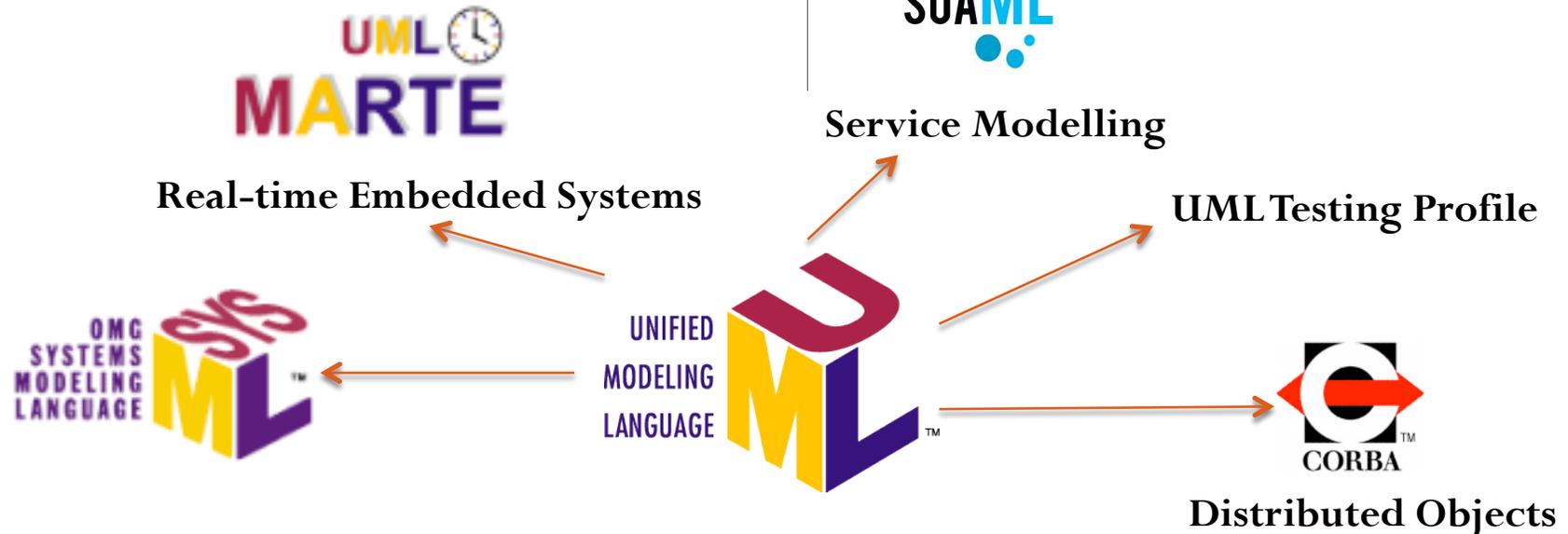
...



Conform to & Use



UML is growing to address domain-specific needs...



Domain specific needs have led to the extension of UML to UML Profiles...

Complexity of UML

Very large meta-model spec. :
246 Classes, 583 properties



Too large and complex to learn and adopt...
Source: Wikipedia on UML



Infact,
many tools need and use
only a **small subset** of
UML !

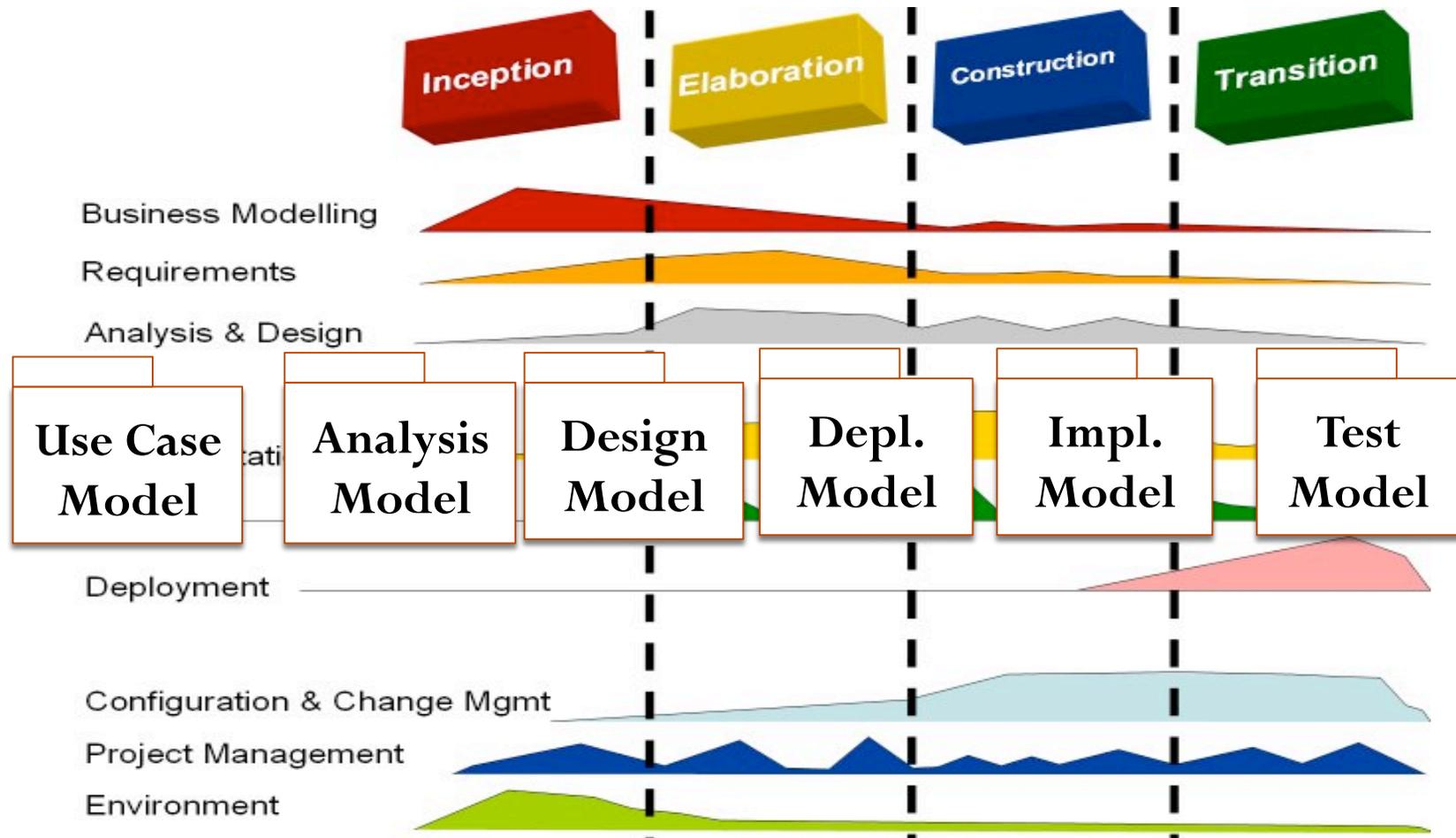
**Not conducive to application
of formal methods...**

Outline

- Where do large meta-models come from?
- **Why a need to extract subsets of large meta-models ?**
- How to prune a meta-model to obtain subsets called effective meta-models ?
- What are the type properties of effective meta-models ?
- Illustrative Case Study : French Space Agency Transformation Input Meta-model
- Conclusion

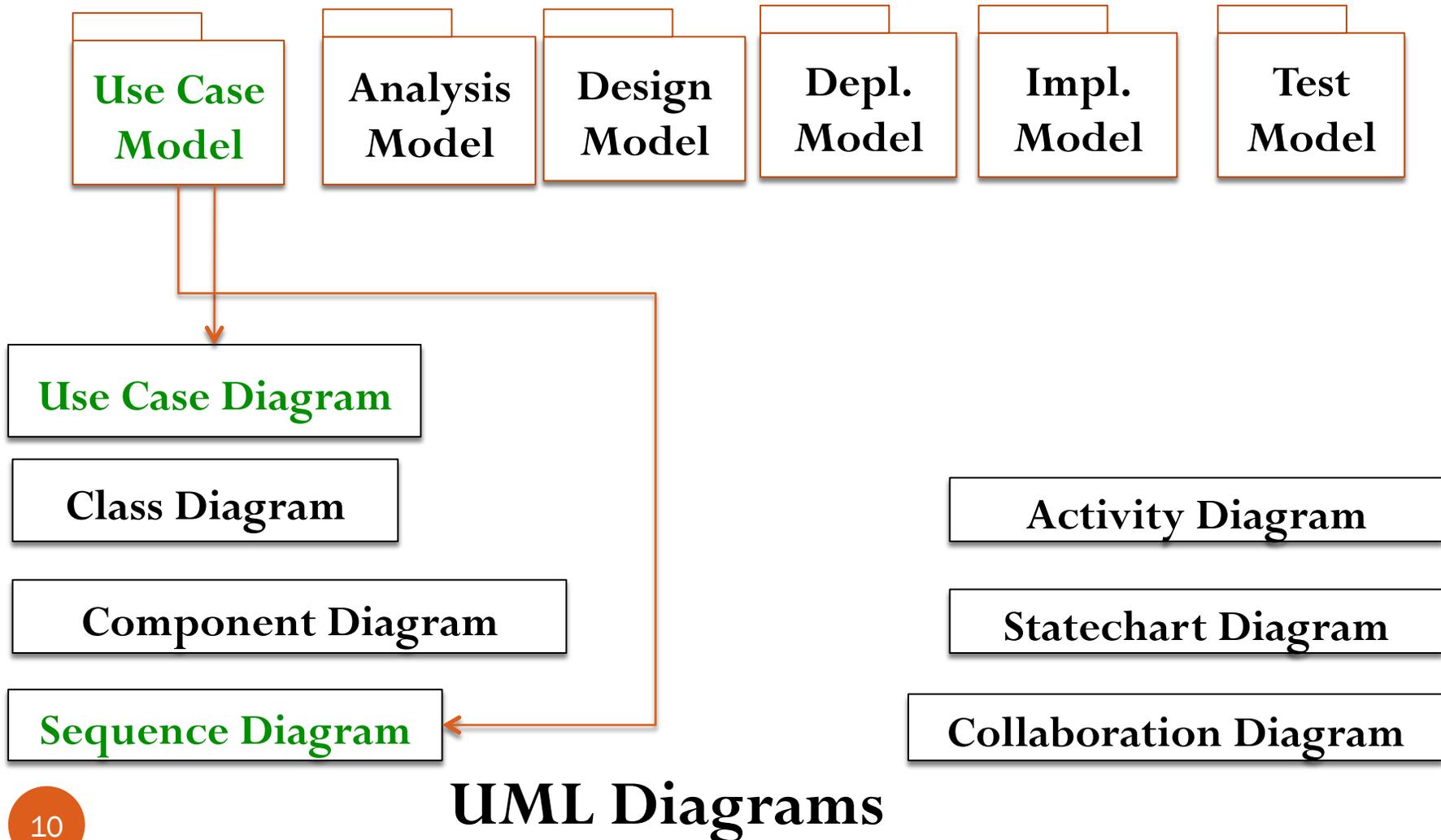


Scenario 1: Why need subsets? UML in Software Process

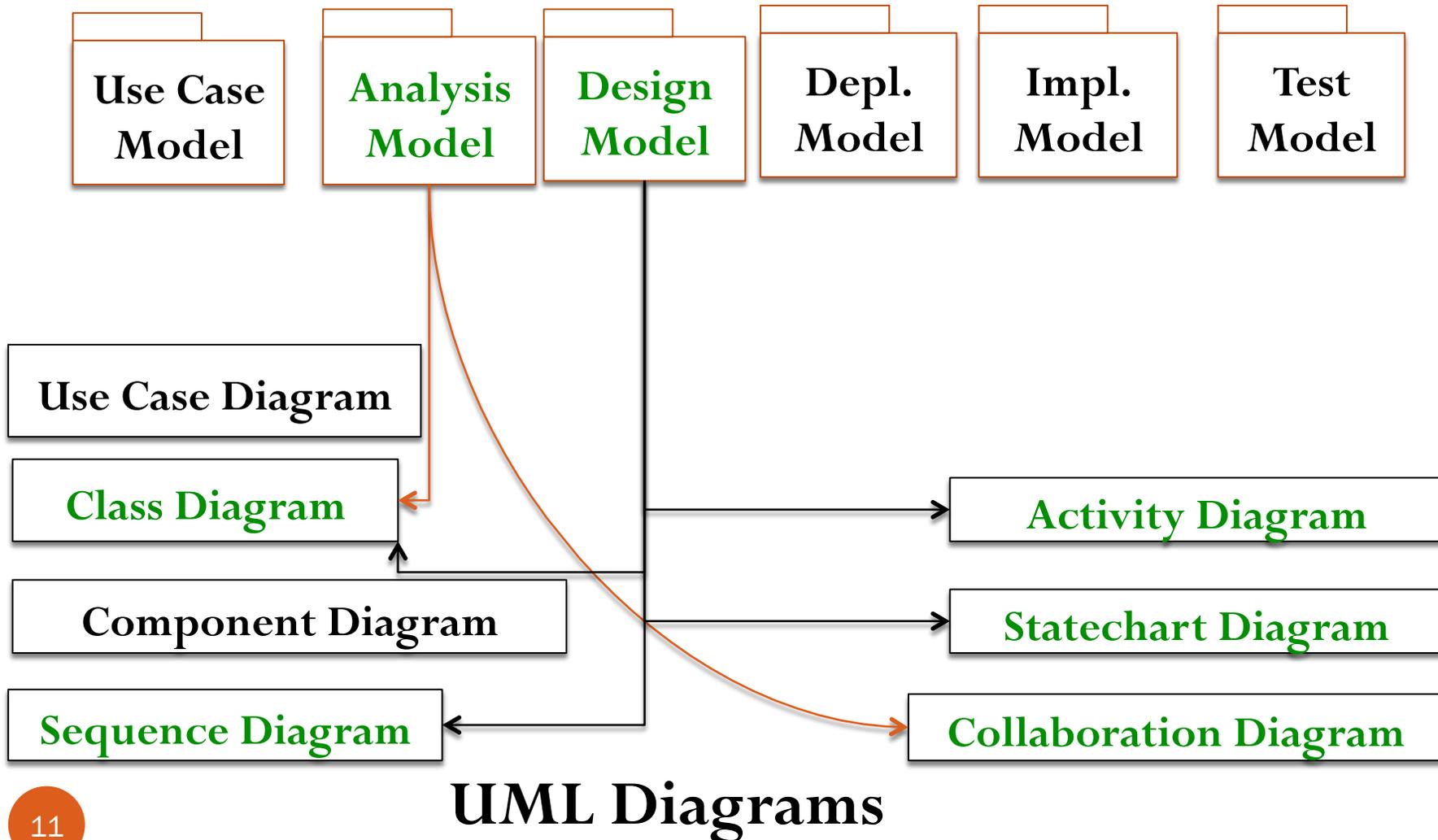


Rational Unified Process

Scenario 1: RUP models via UML



Scenario 1: RUP models via UML

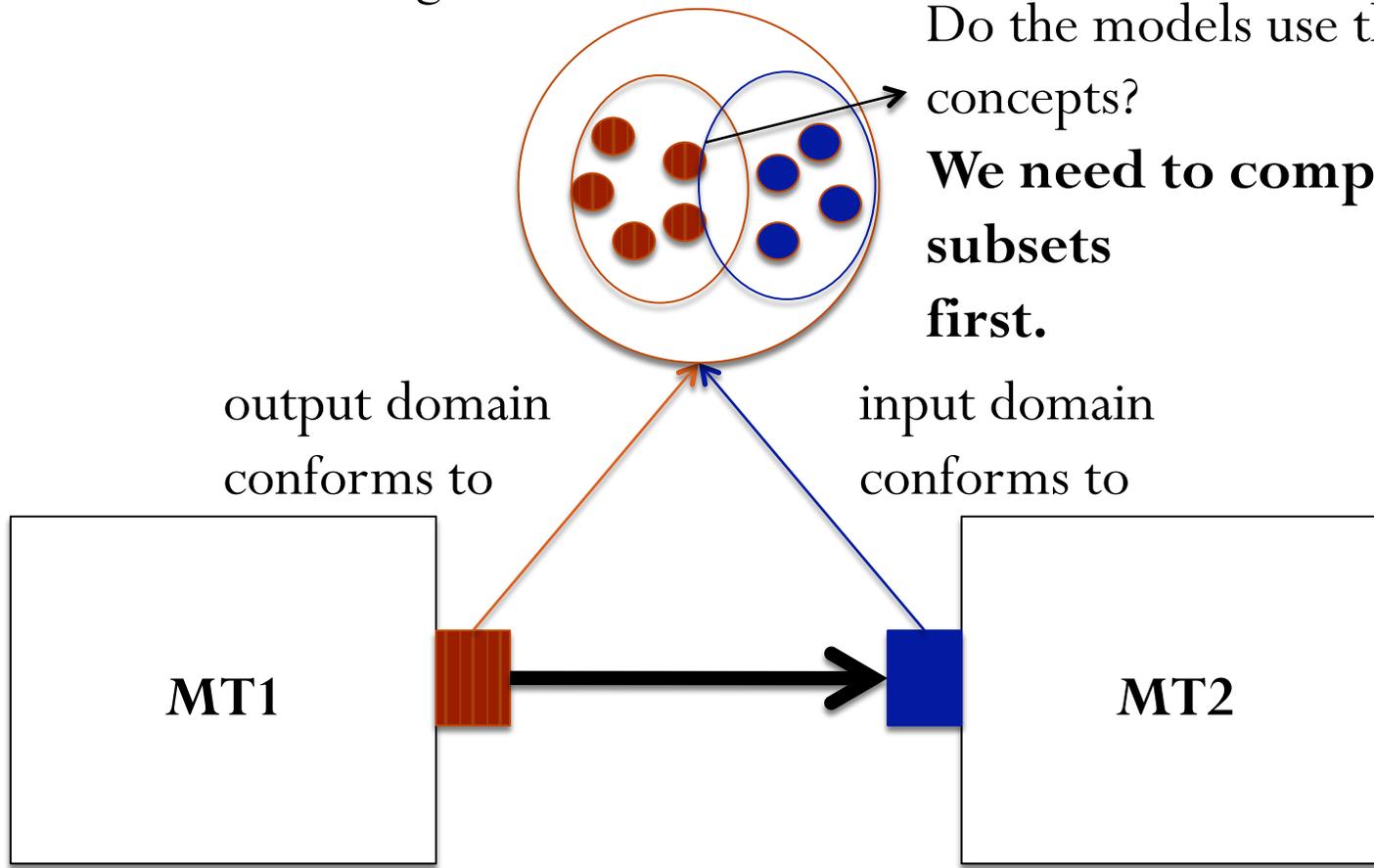


Scenario 2: Why need subsets? Chain of Model Transformations

Large Meta-model
Eg: UML

Are outputs of MT1 valid
inputs for MT2?
Do the models use the same
concepts?

**We need to compare
subsets
first.**



Scenario 3

Existence of hand-made subsets of UML

Google search : “**UML subset**“

You will find at least **1,450** hits (done on Sept. 17,2009)
screaming out the use of UML subsets...

Executable UML

UML subset for FPGA

ICONIX Process: A Core UML subset

UML subset for Model-based Testing

Model checking for executable UML subset
and many more...

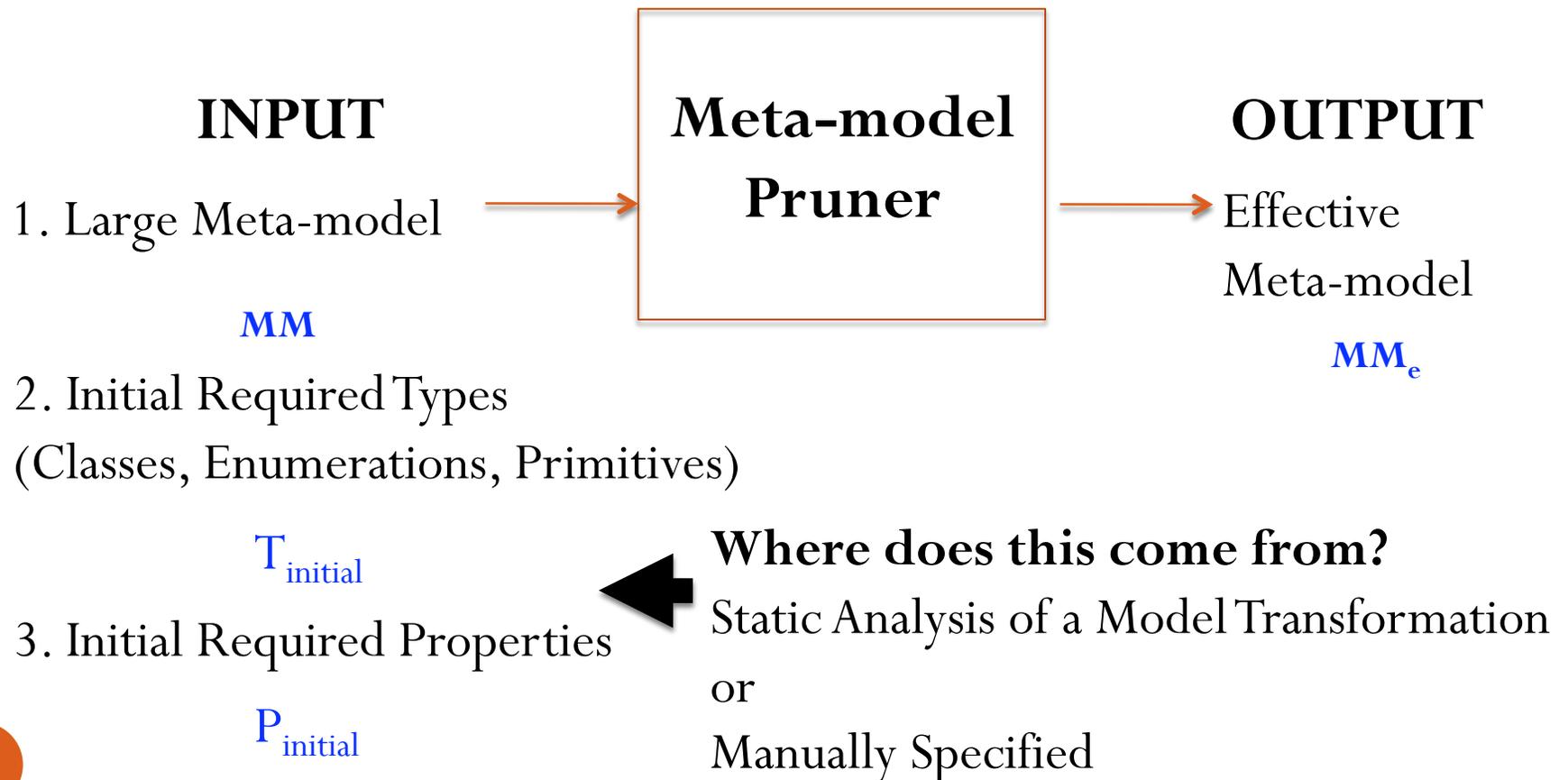


Outline

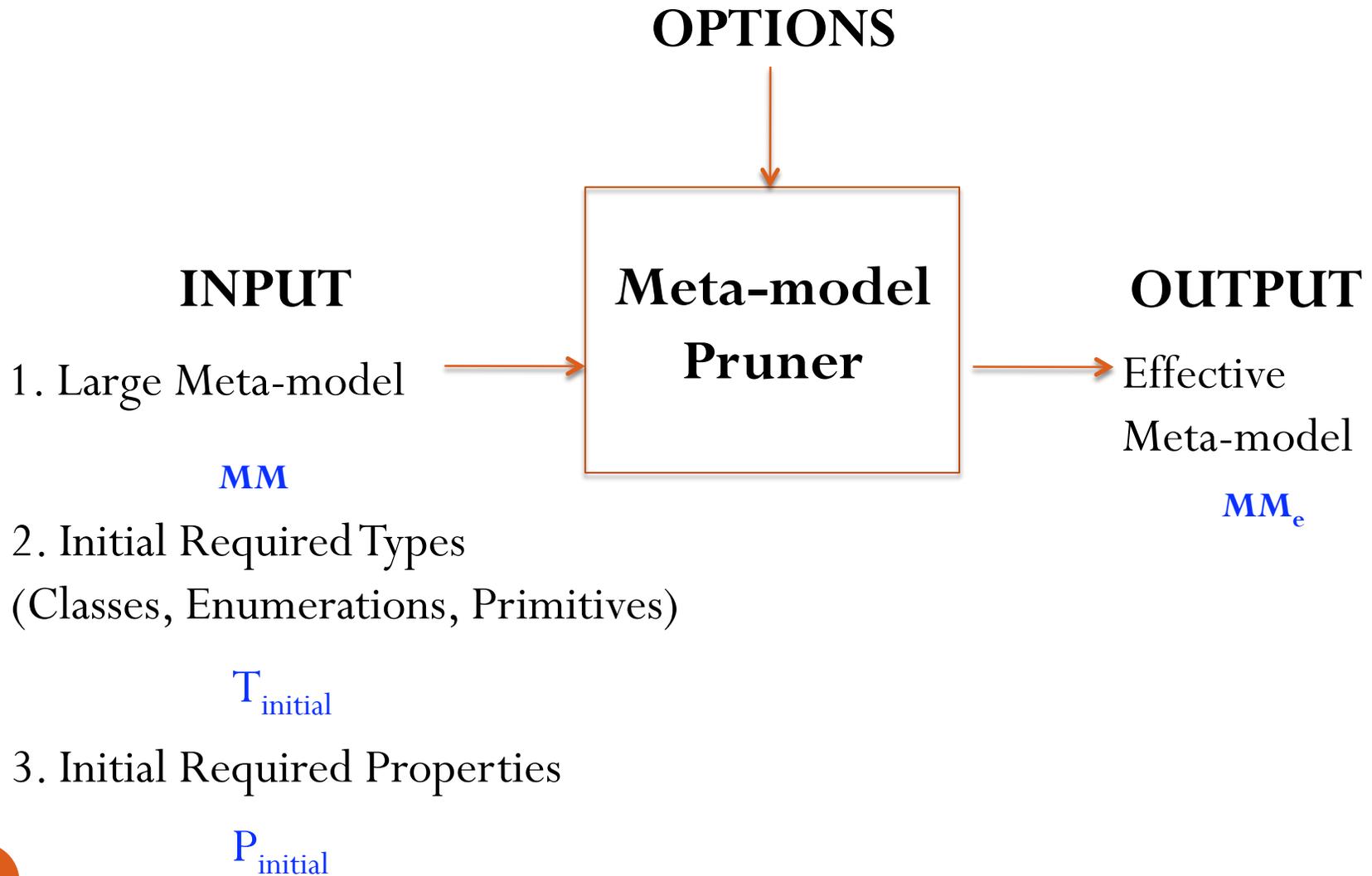
- Where do large meta-models come from?
- Why extract subsets of large meta-models ?
- **How to prune a meta-model to obtain subsets called effective meta-models ?**
- What are the type properties of effective meta-models ?
- Illustrative Case Study : French Space Agency
Transformation Input Meta-model
- Conclusion



Prune based on required concepts



Pruning controlled by options



Meta-model Pruning Internals

**Meta-model
Pruner**

So, what is pruned?

Meta-model Pruning

1. Rules to find set of required types (classes, enumerations, primitives)
2. Rules to find set of required properties
3. Optional rules to add additional types/properties
4. Repeat 1,2,3 until **rules** cannot be applied anymore

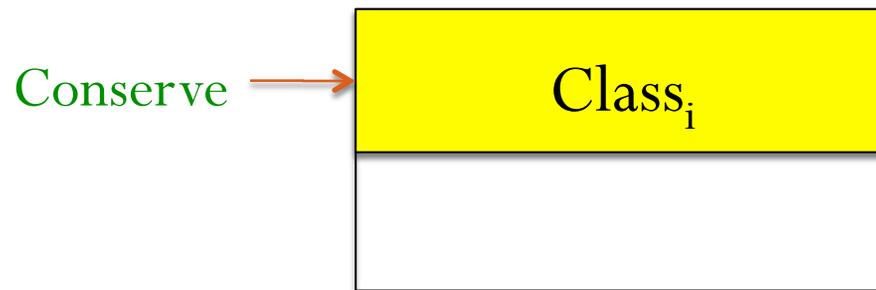
Lets look at some of these rules...

Meta-model Pruning

Couple of rules to conserve types...

Finding Required Types

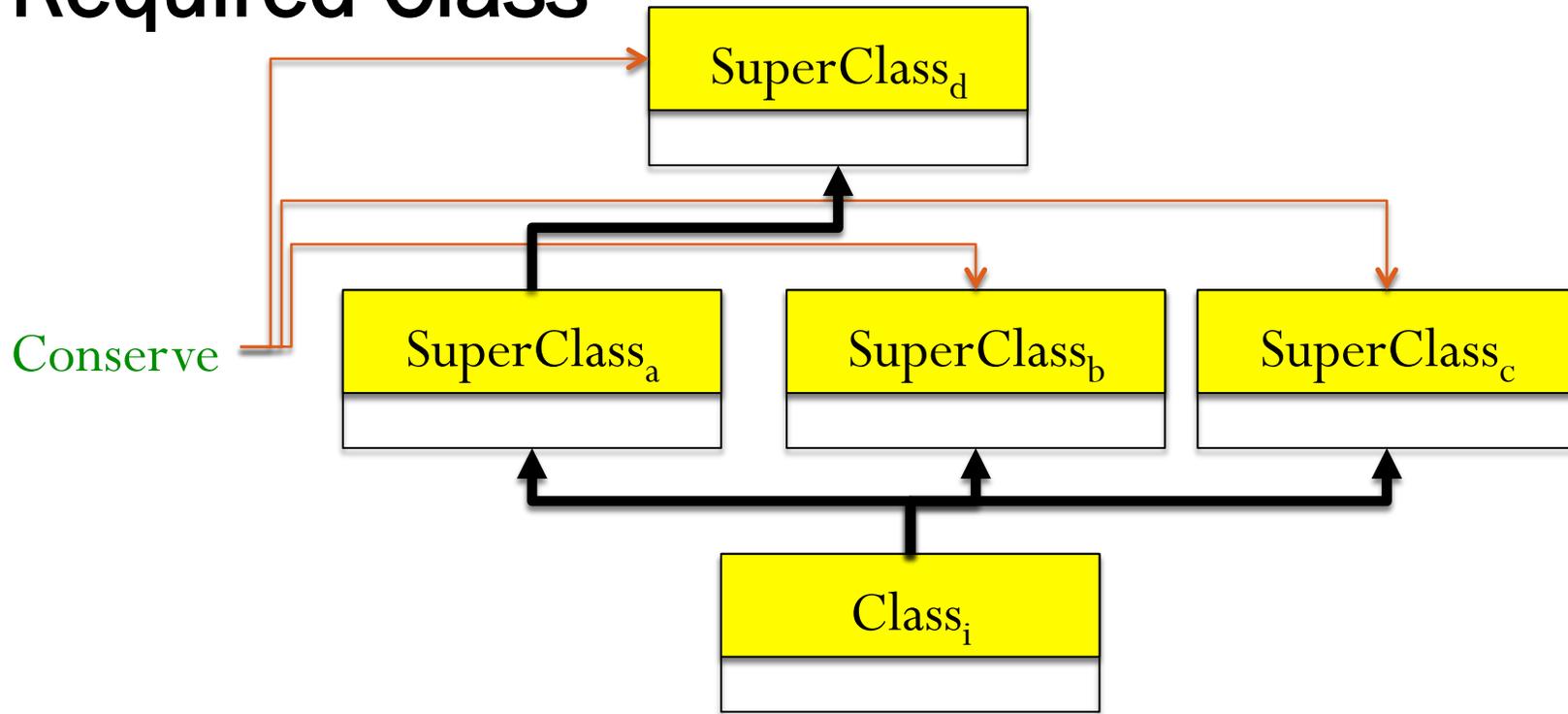
Rule: Conserve Initial Required Class



$$T_{\text{req}} \leftarrow T_{\text{req}} + \text{Class}_i, \text{ if } \text{Class}_i \text{ in } T_{\text{initial}}$$

Finding Required Types

Rule: Conserve All Super Classes of Required Class



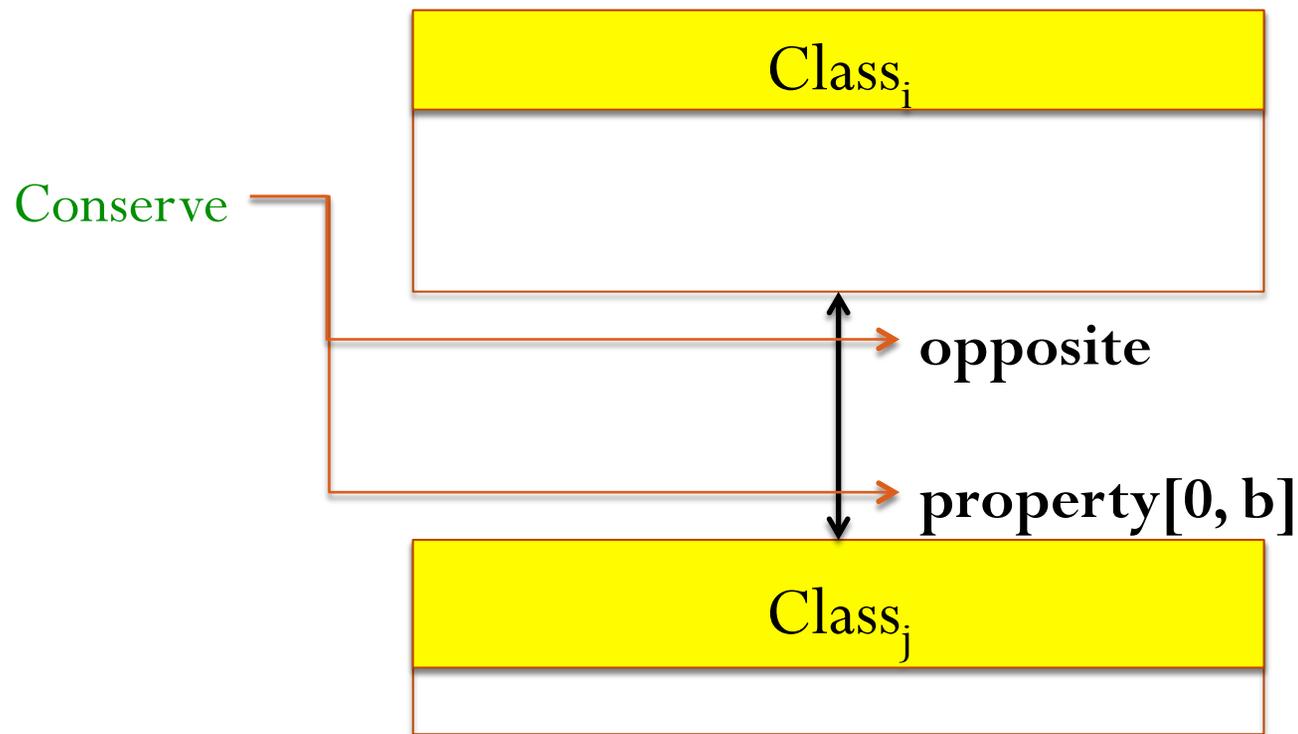
$$T_{\text{req}} \leftarrow T_{\text{req}} + \text{SuperClass}_i, \text{ if } \text{Class}_i \text{ is derived from } \text{SuperClass}_i$$

Meta-model Pruning

Couple of rules to conserve properties...

Finding Required Properties

Rule: Conserve Property of Required Type

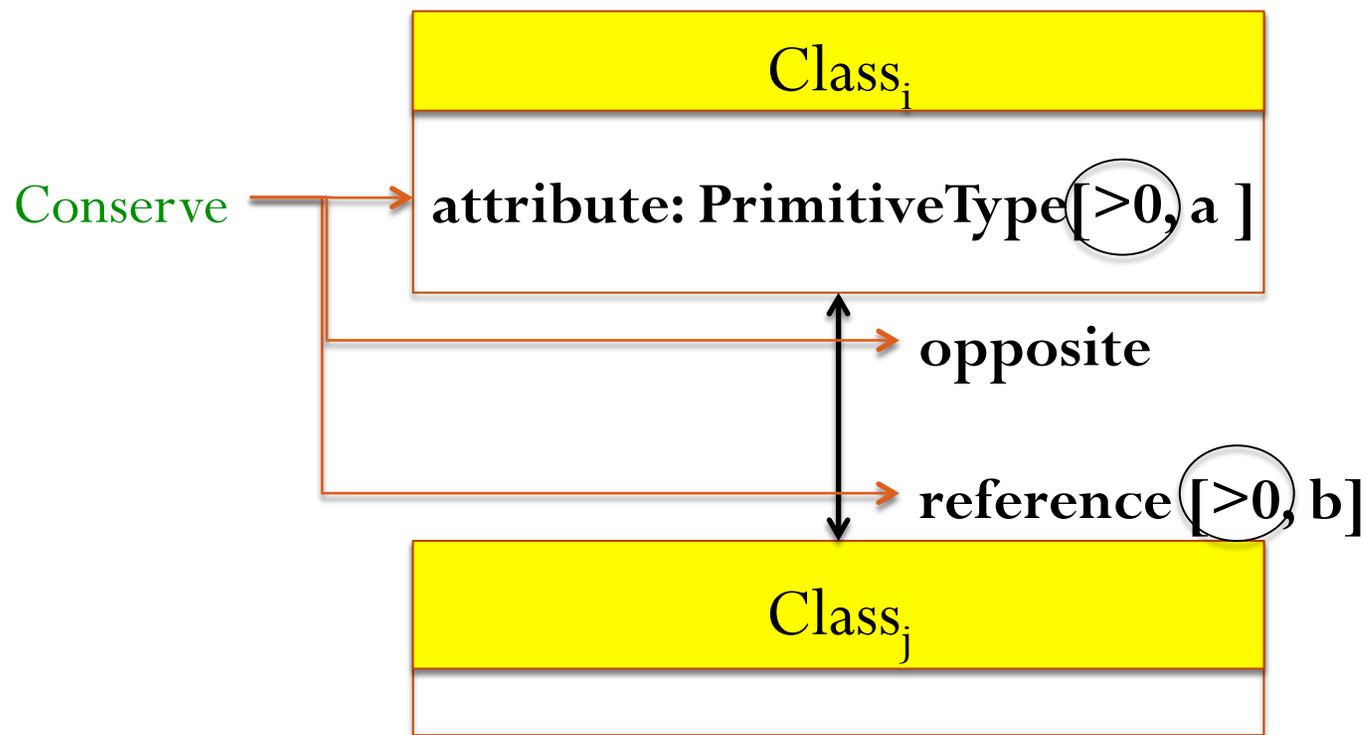


if property.type = Class_j in T_{req} $P_{req} \leftarrow P_{req} + \text{property}$

$P_{req} \leftarrow P_{req} + \text{opposite}$

Finding Required Properties

Rule: Conserve Multiplicity Property



if $\text{property.lower} > 0$ and Class_i in T_{req} , $P_{\text{req}} \leftarrow P_{\text{req}} + \text{property}$

24

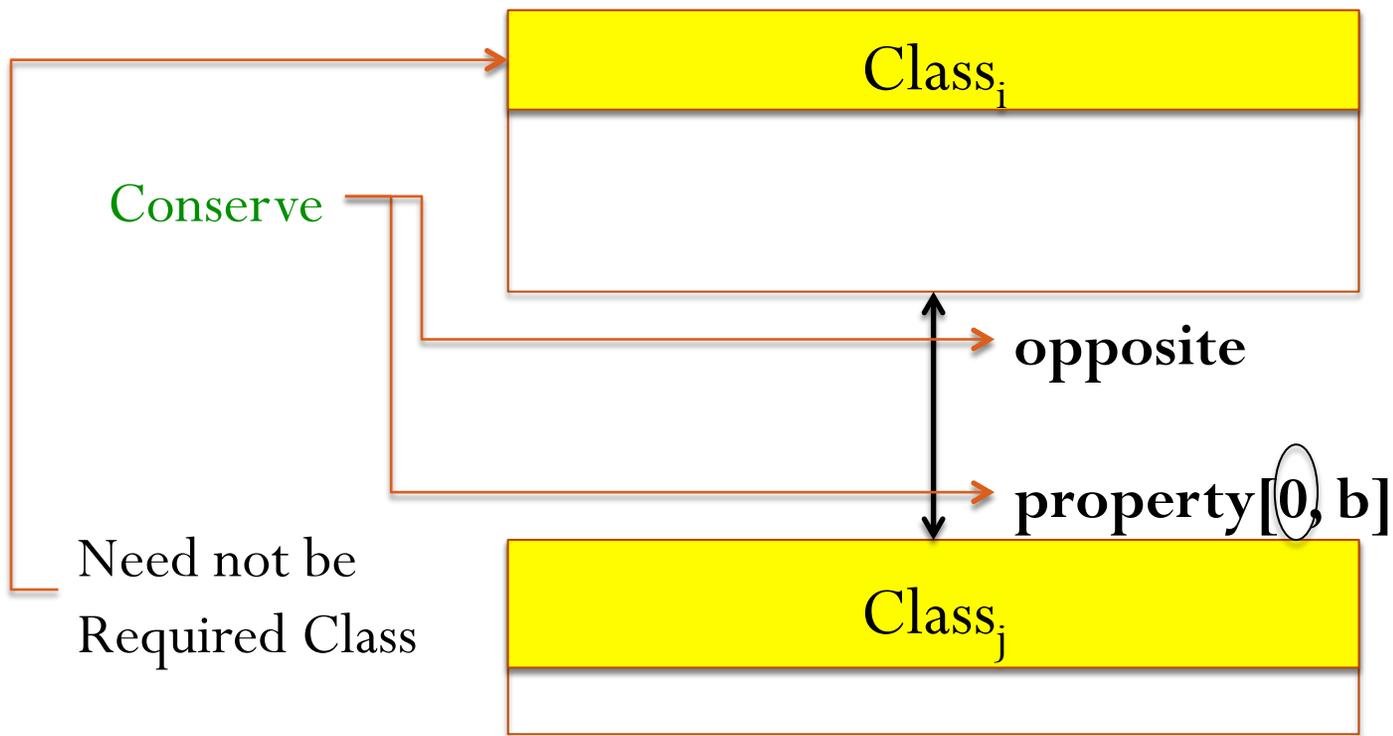
$T_{\text{req}} \leftarrow T_{\text{req}} + \text{Class}_j$

$P_{\text{req}} \leftarrow P_{\text{req}} + \text{opposite}$

Meta-model Pruning

A rule to illustrate options...

Optional Rule: Conserve Zero Multiplicity Property



if `property.lower == 0` and
`property.type.isInstanceOf(Class)`,

$$P_{req} \leftarrow P_{req} + \text{property}$$

$$P_{req} \leftarrow P_{req} + \text{opposite}$$

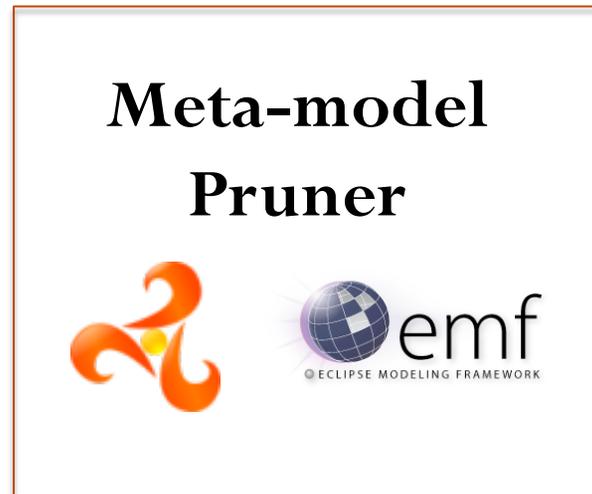
Meta-model Pruning

Apply rules (about 16) until no rule can be applied anymore...

Meta-model Pruning

Remove types and properties not conserved...

Meta-model Pruning Implementation



- ◆ Implemented in Kermeta (<http://www.kermeta.org>)
- ◆ Conforms to the EMF Ecore file format for input
- ◆ Download at:

<http://www.irisa.fr/triskell/software-fr/protos/metamodelpruner/>

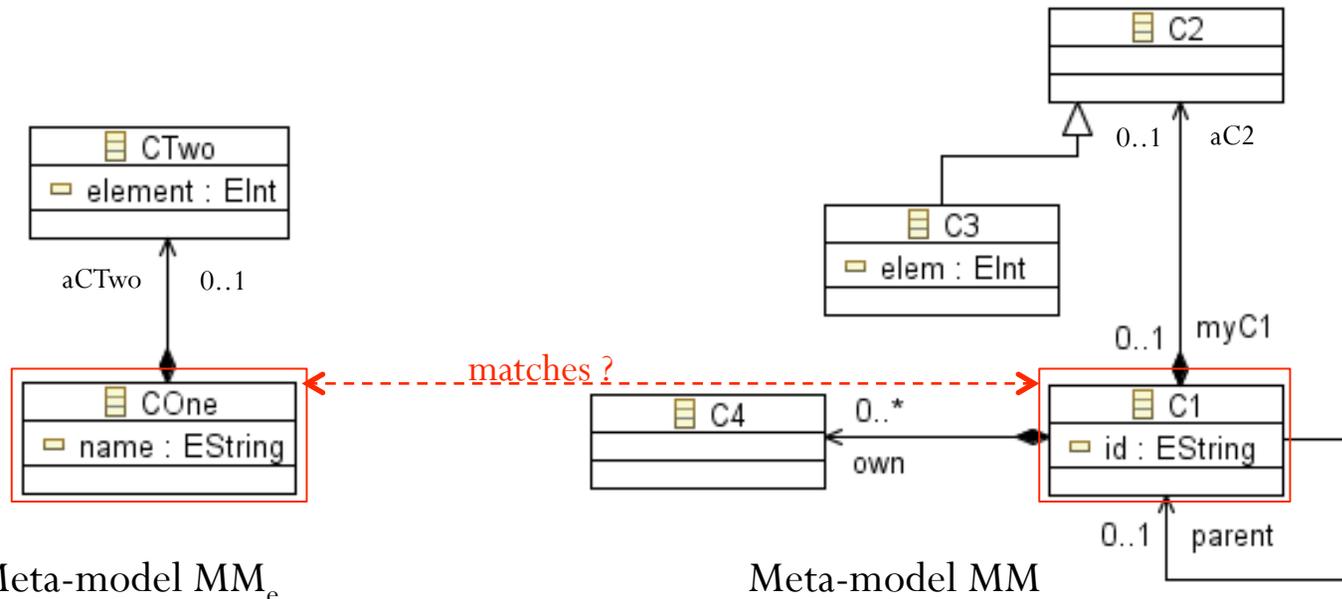
Outline

- Where do large meta-models come from?
- Why extract subsets of large meta-models ?
- How to prune a meta-model to obtain subsets called effective meta-models ?
- **What are the type properties of an effective meta-model ?**
- Illustrative Case Study : French Space Agency Transformation Input Meta-model
- Conclusion



Model Typing

- Does the effective meta-model MM_e match the large meta-model MM ?



- Meta-model MM_e **matches** another meta-model MM **iff** for each **class C in MM** , there is one and only one **corresponding class C'** in MM_e
- Set of rules guarantees a **subtype relationship** btw the two MM s [Steel 07, SoSyM]
- If transformation T works for MM , and if MM_e is a model subtype of MM , then T works with MM_e

What are properties of an effective meta-model?

- We verify that an effective meta-model is *super-type* of the large meta-model using *model typing*.
- This implies all *transformations written* for the effective meta-model are *also valid for* the large meta-model
- *All instances* (models) of the effective meta-model are *also instances of the* large meta-model.
- The instances are compatible with the large meta-model because *we preserve concept names*.
- *The effective meta-model can live its own life much like a DSML!*

Outline

- Where do large meta-models come from?
- Why extract subsets of large meta-models ?
- How to prune a meta-model to obtain subsets called effective meta-models ?
- What are the type properties of an effective meta-model ?
- **Illustrative Case Study : French Space Agency Transformation Input Meta-model**
- Conclusion



French Space Agency Transformation Case Study

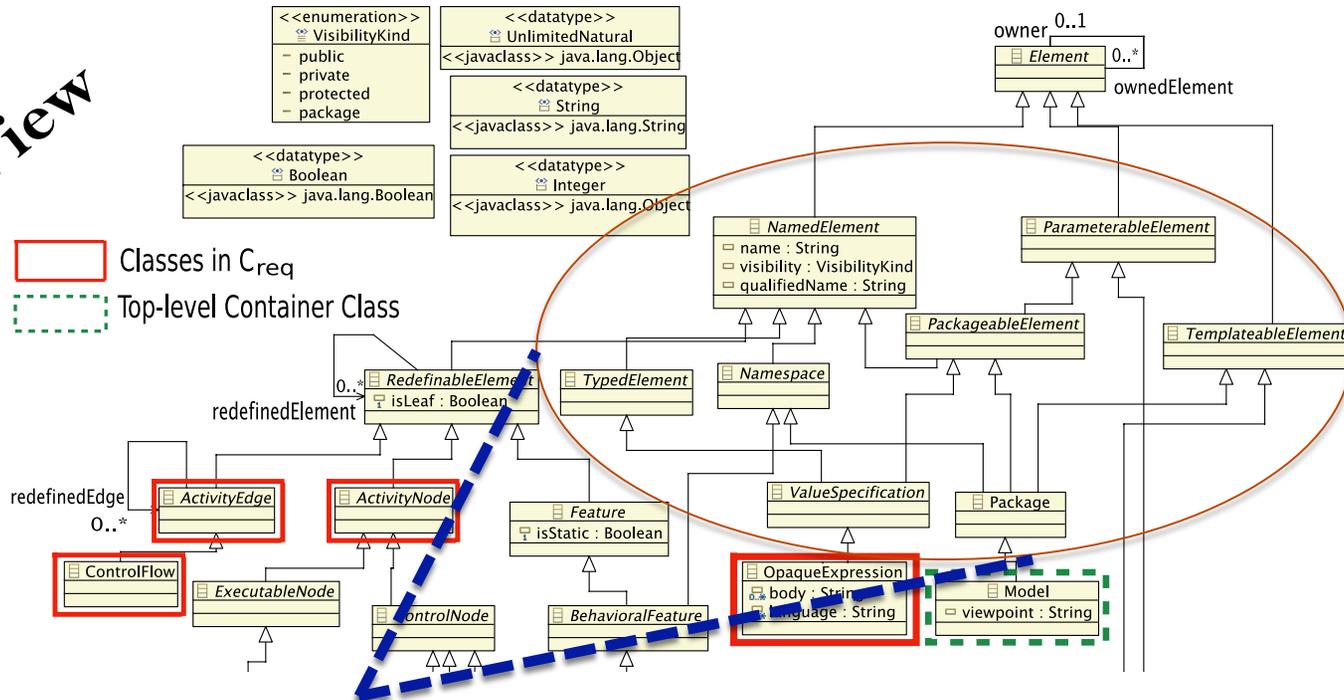
- Transformation for generation embedded system code from a subset of UML Activity Diagrams
- **Transformation analysis** : Gives a set of required classes and properties in the UML
- **Meta-model pruning**: Gives an output effective meta-model which is a super type and subset of UML.

Set of Required Classes in UML (Total = 10)

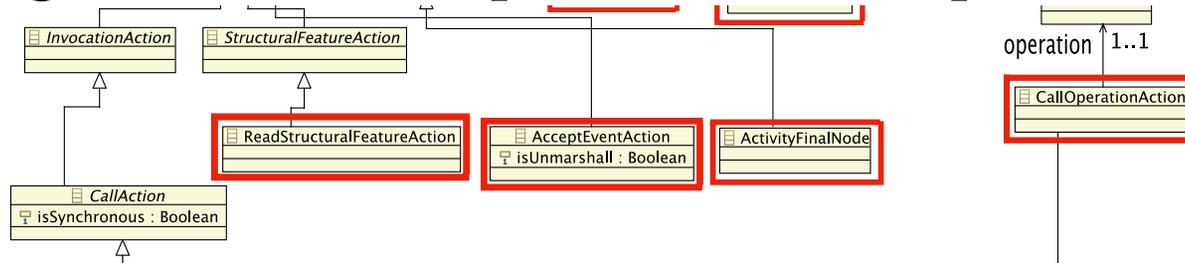
- ActivityEdge
- ActivityNode
- ControlFlow
- OpaqueExpression
- DecisionNode
- InitialNode
- ReadStructuralFeatureAction
- AcceptEventAction
- ActivityFinalNode
- CallOperationAction

Effective UML for Case Study

Bird's Eye View



Using Conserve All Super Classes of Required Class



Observations

1. A significant reduction in size of the input meta-model for the transformation:

31 Classes and 15 Properties Vs. 246 Classes and 583 Properties

2. We obtain the subset of UML that is truly transformed. **This minimizes errors in model construction.**

3. The Effective UML meta-model is **a super type of UML.** (We verify this in the impl.)

4. All instances of operations of the effective UML meta-model are compatible with UML itself.

Outline

- Where do large meta-models come from?
- Why extract subsets of large meta-models ?
- How to prune a meta-model to obtain subsets called effective meta-models ?
- What are the type properties of effective meta-models ?
- Illustrative Case Study : French Space Agency
Transformation Input Meta-model
- **Conclusion**



Conclusion

- We present an important problem associated with using large meta-models such as the UML
- We show how to **work around** the problem by extracting a smaller effective meta-model via **meta-model pruning**.
- We show that that the **effective meta-model is a super type** of the original large meta-model.
- **All instances and operations** of the effective meta-model are **backward compatible** with the large meta-model.
- We demonstrate our approach to obtain the effective meta-model on **an industrial case study**.
- Often, there is a **significant difference in sizes** of the effective and the original large meta-model!

Future Work

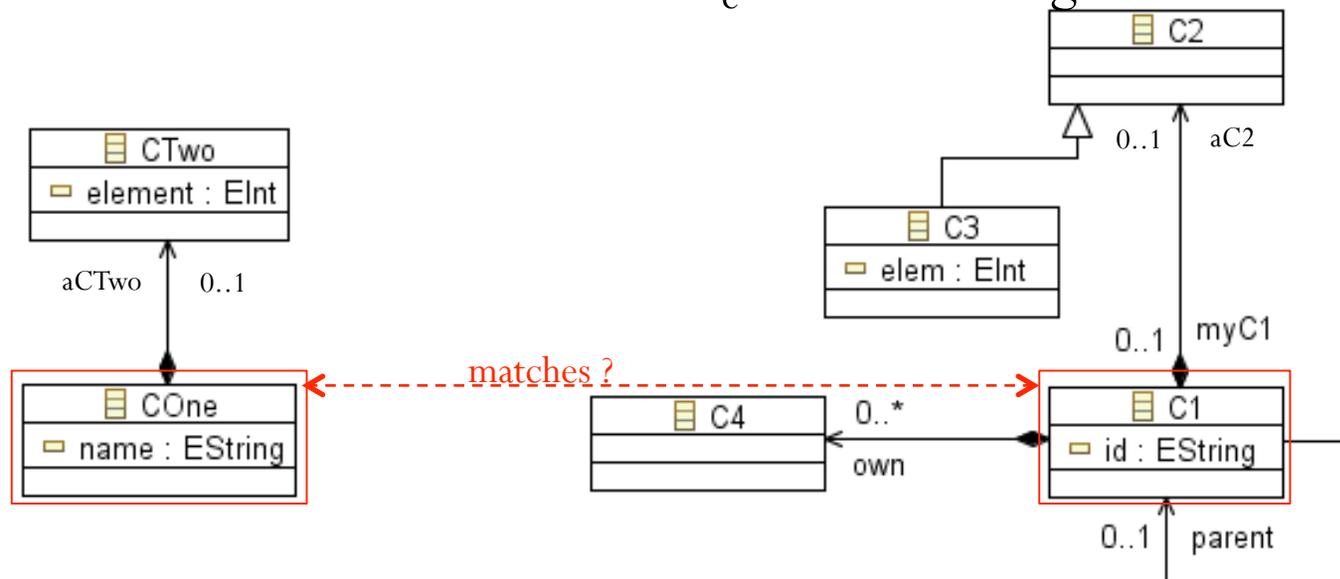
- **Atomic pruning operators** that generate super-types of input meta-model
- **Any sequence of these operators** lead to a super-type **by construction**. Thanks to **transitivity**.
- For now, **we verify** that the result is a super-type using model typing.
- Meta-model pruning demonstrates the possibility for a **whole family of algorithms**. Ours is just an example.
- A new pruning operator could possibly deal with **flattening the meta-model inheritance hierarchy**

Thank You!

Annexe : Model Typing

Model Typing : Type properties of MM_e

- Does the effective meta-model MM_e match the large meta-model MM ?



context **Class** def:

```
localMatchFor ( c : Class ) : Boolean =
```

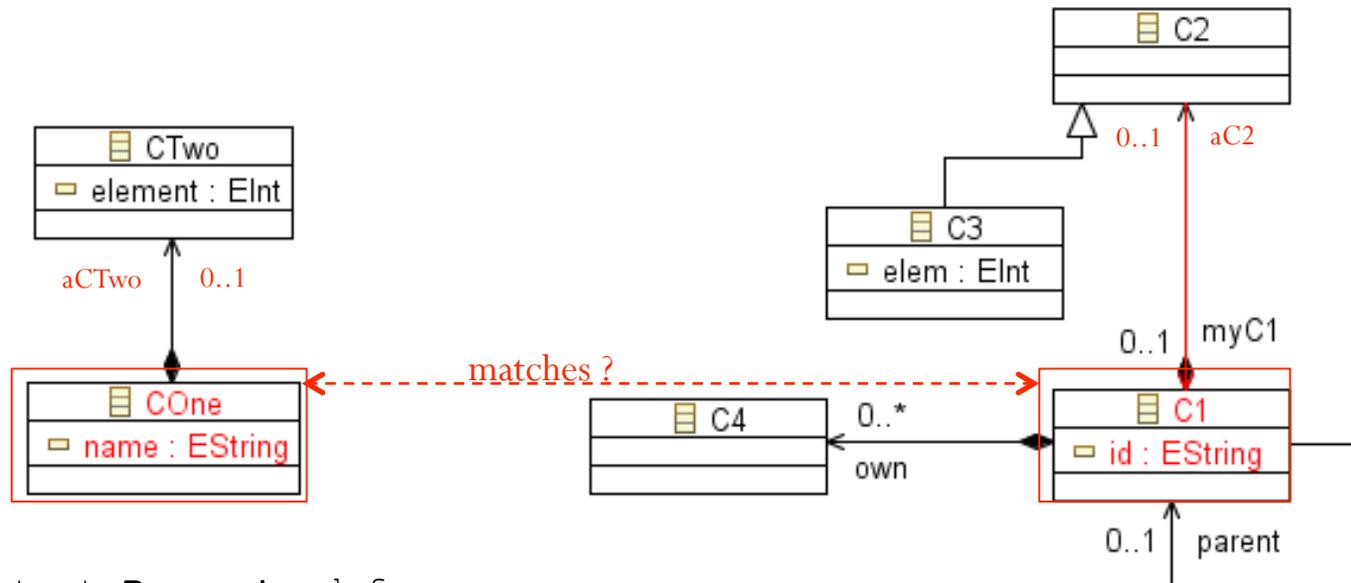
```
  not c.isAbstract implies not self.isAbstract
```

```
  and c.ownedProperty -> forAll ( p |
    self.ownedProperty -> exists ( p2 | p2.localMatchFor (p) ) )
```

```
  and c.ownedOperation -> forAll ( o |
    self.ownedOperation -> exists ( o2 | o2.localMatchFor (o) ) )
```

Model Typing

- Does the effective meta-model MM_e match the large meta-model MM ?



context **Property** def:

localMatchFor (**p** : **Property**) : Boolean =

~~self.name = p.name~~

and **self.multiplicityMatchFor**(p)

and **not p.isReadOnly** implies **not self.isReadOnly**

and **self.isComposite** = **p.isComposite**

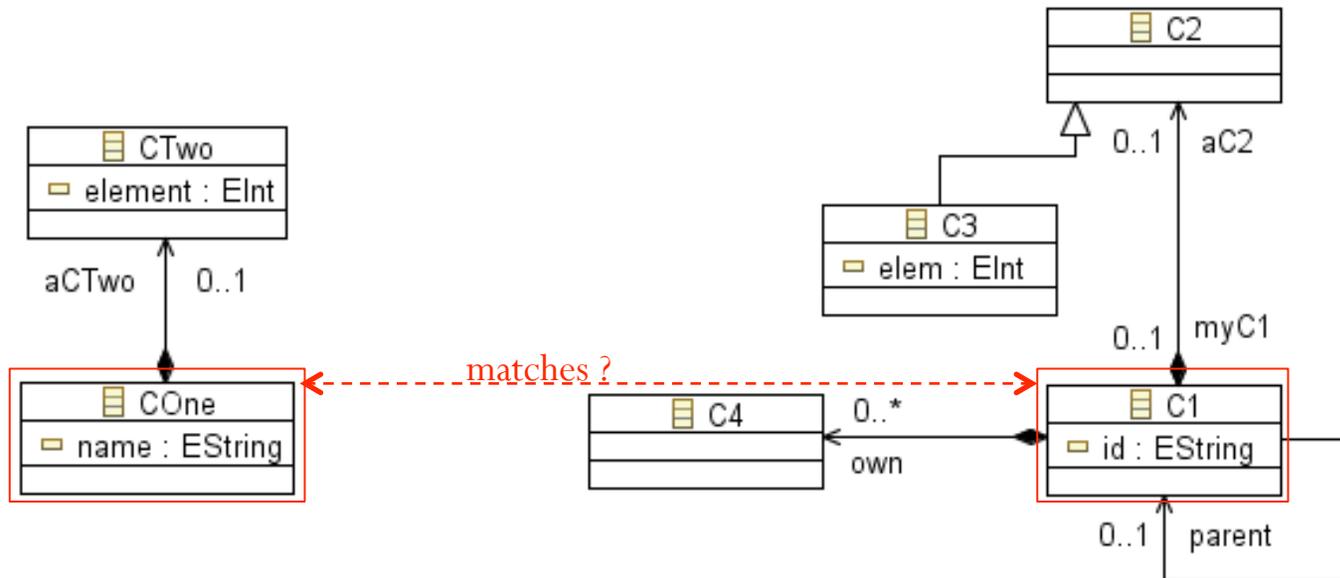
and not p.opposite -> isOclUndefined implies

not self.opposite -> isOclUndefined

and (~~self.opposite.name = p.opposite.name~~)

Model Typing

- Does the effective meta-model MM_e match the large meta-model MM ?



```
context MultiplicityElement
multiplicityMatchFor ( m : MultiplicityElement ) : Boolean =
    (m.upper = 1) implies (self.upper = 1)
    and self.upper <= m.upper
    and self.lower >= m.lower
    and m.isOrdered implies self.isOrdered
    and self.isUnique = m.isUnique
```