FLEXIBLE MODELLING BASED ON FACETS

Juan de Lara[1]

Joint work with
E. Guerra[1], J. Kienzle[2], Y. Hattab[2]

[1]Universidad Autónoma de Madrid (Spain)
[2]McGill University (Canada)

UAM UNIVERSIDAD AUTONOMA DE MADRID

McGill

AGENDA

Context: Model-driven Engineering

Motivation

Facets

Interfaces and laws

Tool support

Case studies

Conclusions and future work

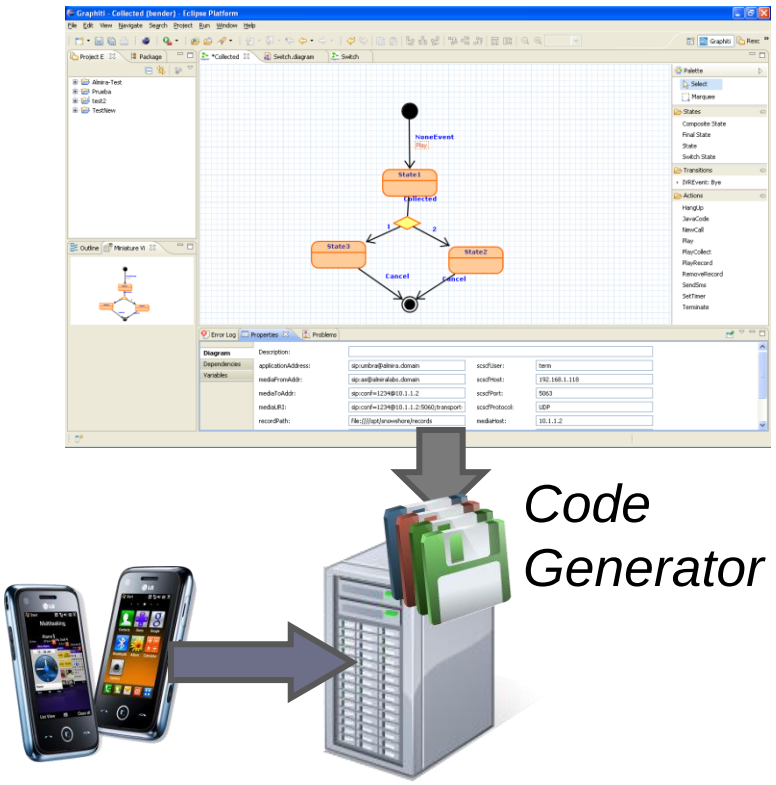
CONTEXT: MODEL-DRIVEN ENGINEERING

For specific domains

- Avoid coding the same solutions over and over
- Families of applications

Domain-Specific Modelling Languages (DSLs)

- Syntax defined by a meta-model
- Code generators, simulators, model transformations



Modelling, validation and automatic generation of telephony services

MDE: DOMAIN-SPECIFIC LANGUAGES

Abstract syntax
- Meta-model
- Integrity constraints (OCL)

Concrete syntax
- Graphical
- Textual
- Tabular, etc

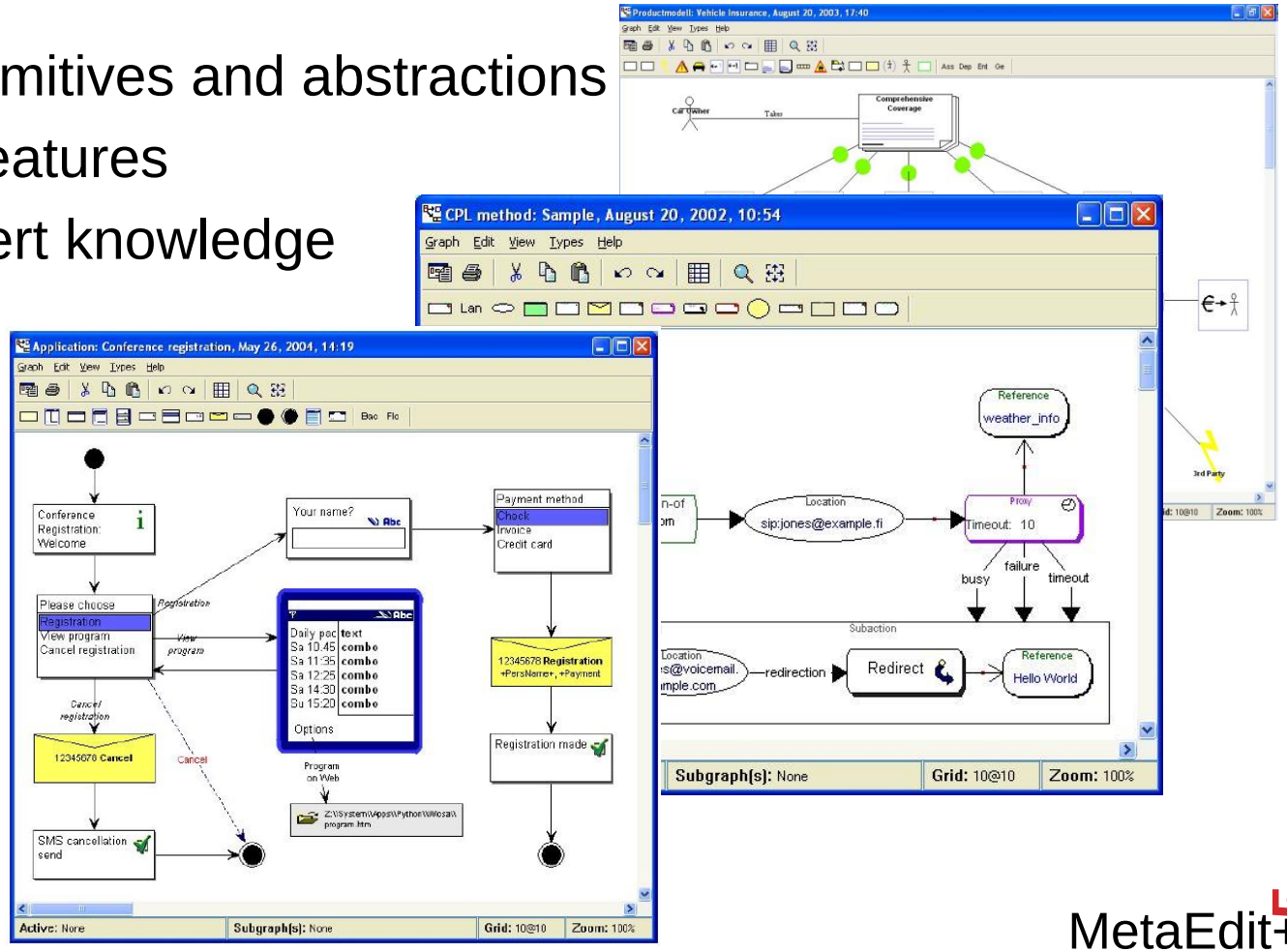
Semantics
- Operational, denotational, etc

Services
- Refactorings, code generators, simulators, etc

DOMAIN SPECIFIC MODELLING LANGUAGES

Describe the structure of the domain

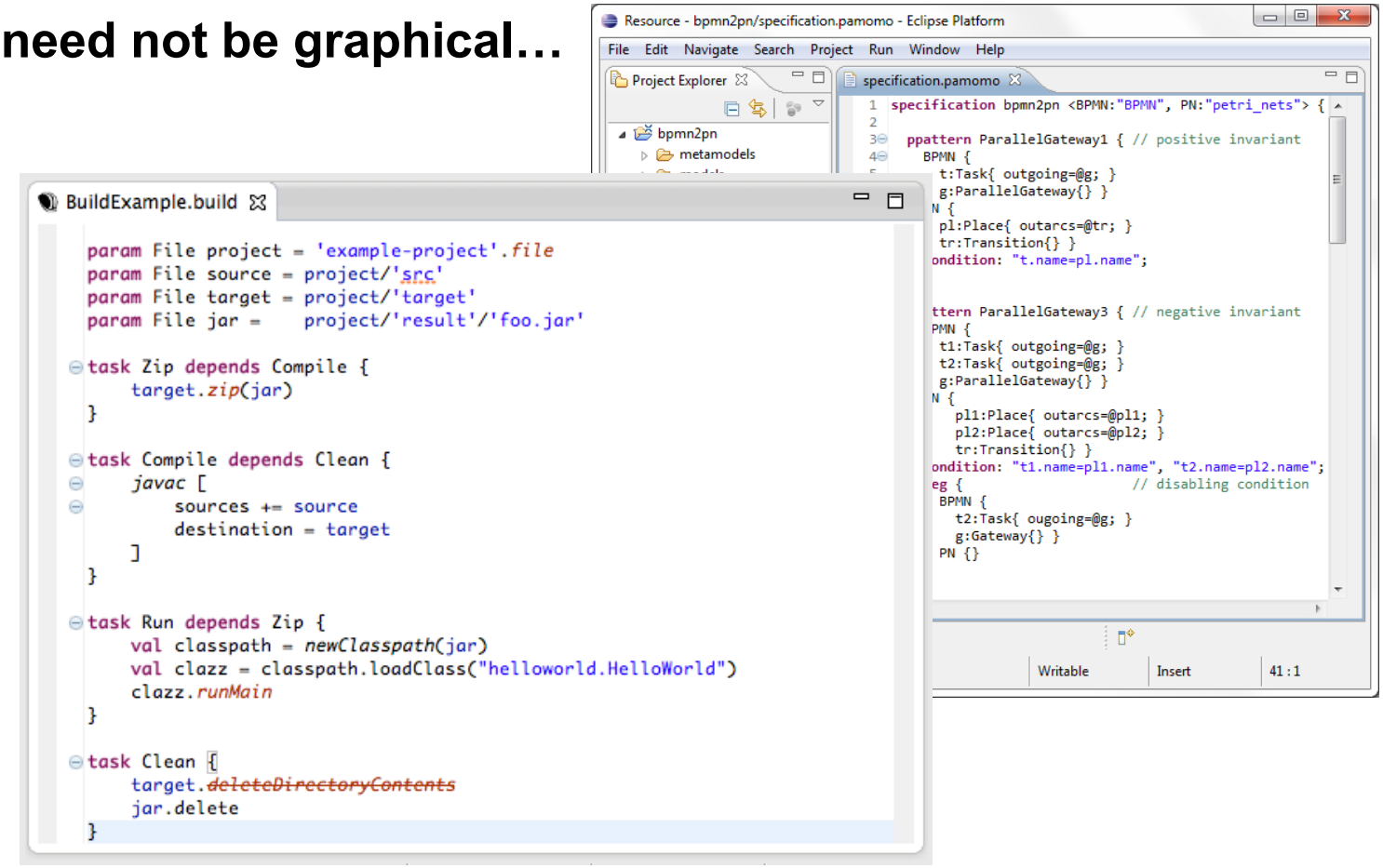
- Relevant primitives and abstractions
- Relations, features
- Explicit expert knowledge



MetaEdit+

DOMAIN SPECIFIC MODELLING LANGUAGES

DSMLs need not be graphical…

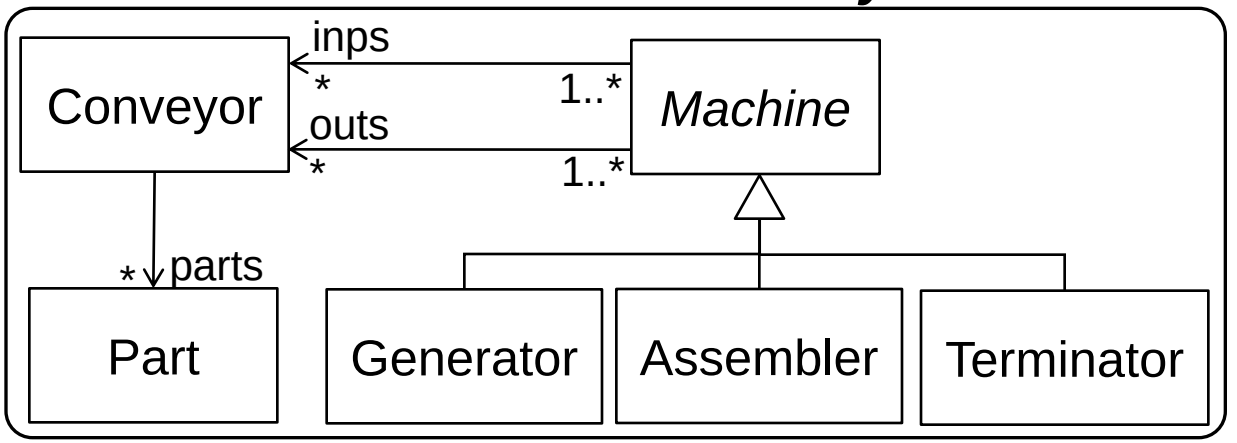


MODELS AND META-MODELS

The abstract syntax of DSMLs is defined through a meta-model

- Classes
- Attributes
- Relations

Factory meta-model



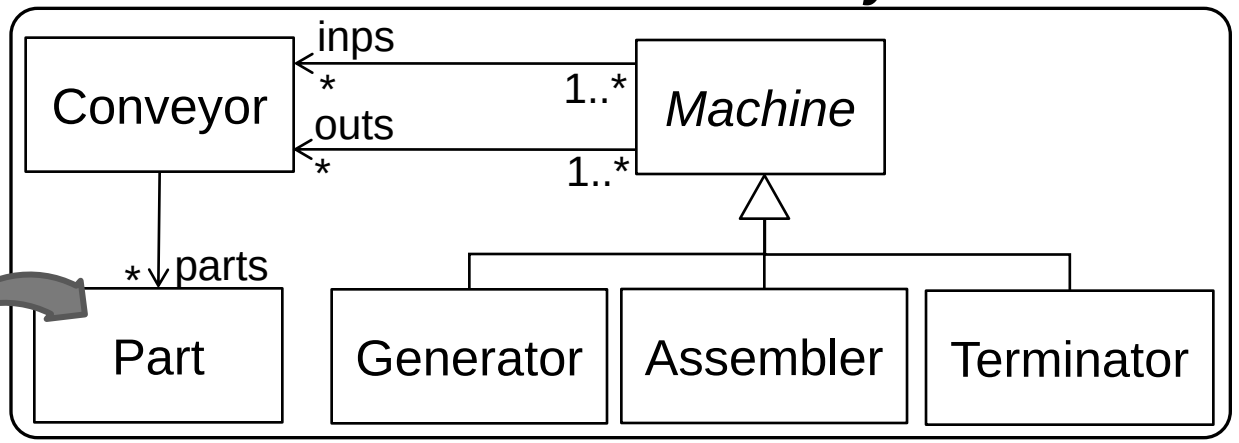
MODELS AND META-MODELS

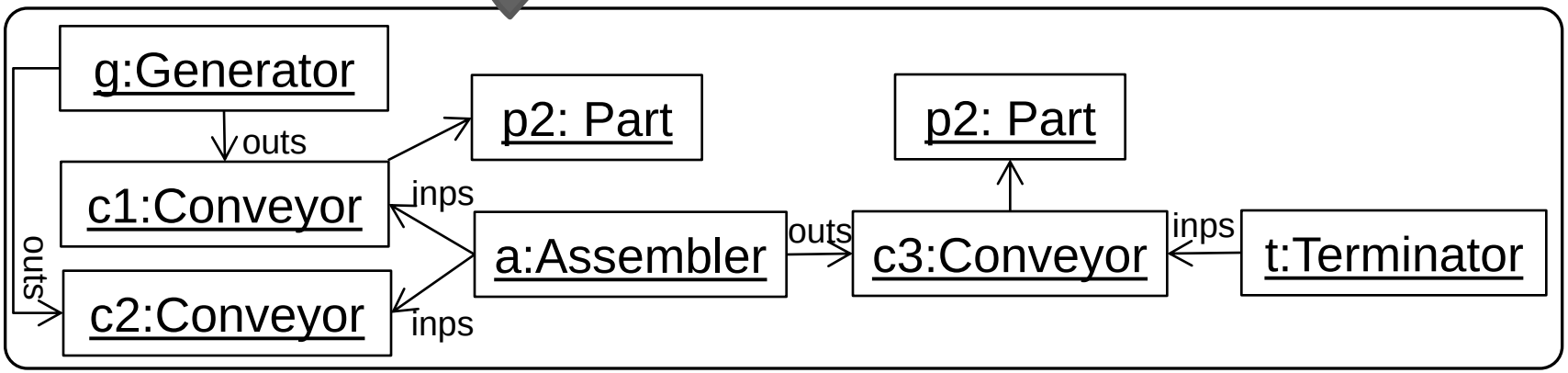
The abstract syntax of DSMLs is defined through a meta-model

- Classes
- Attributes
- Relations

«conforms to»

Factory meta-model





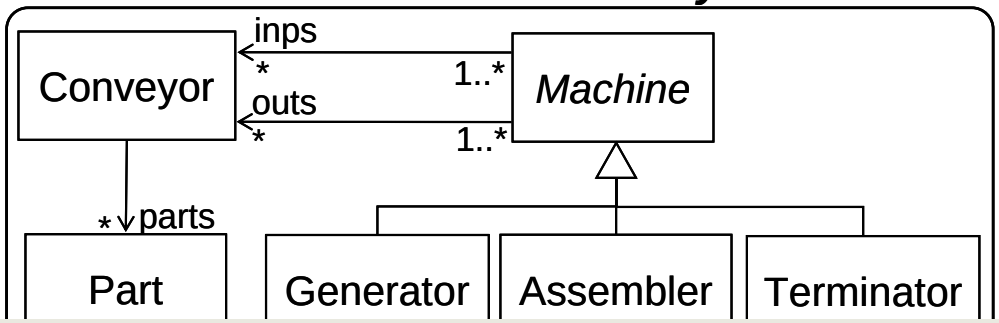
OCL CONSTRAINTS

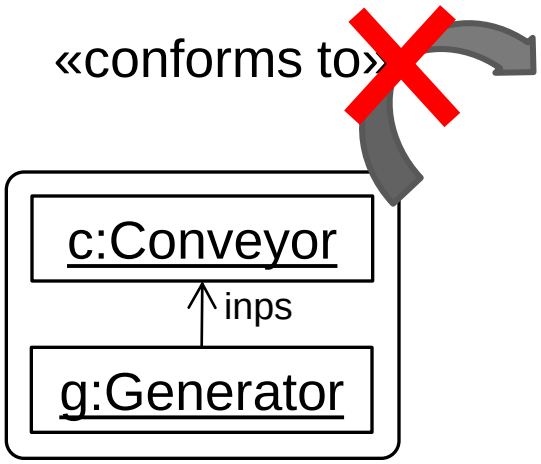
Object Constraint Language

Well-formedness rules, which every model should satisfy

Based on First-Order Logic

Factory meta-model



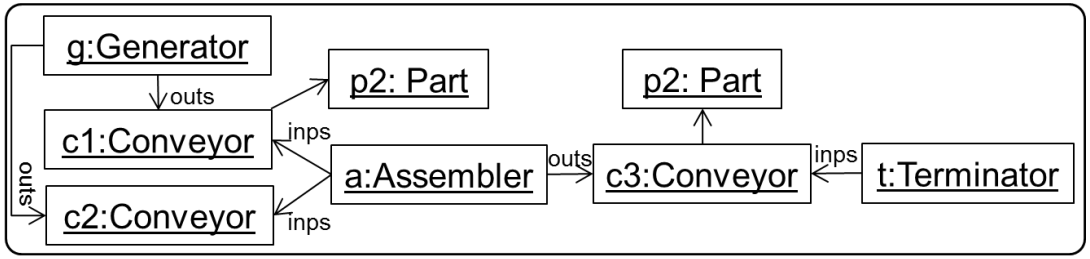


context Generator **inv**:
self.inps->isEmpty() **and self**.outs->size()>0
context Terminator **inv**:
self.outs->isEmpty() **and self**.inps->size()>0
context Assembler **inv**:
self.inps->size()>0 **and self**.outs.size()>0
…

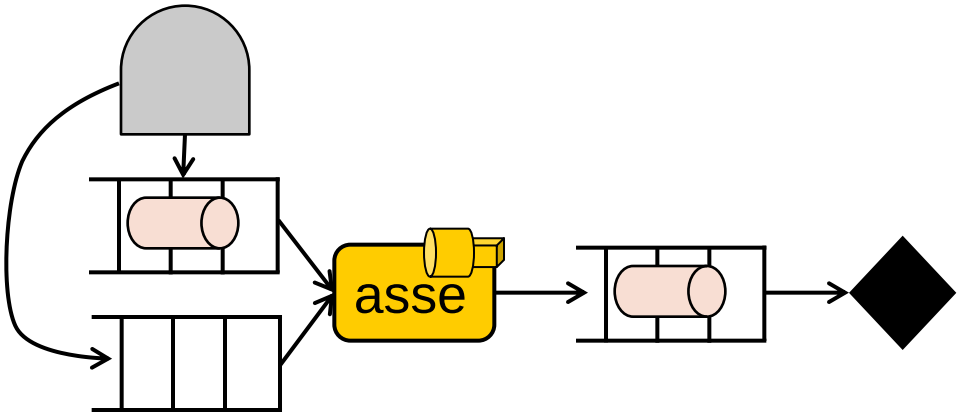
MODELS AND META-MODELS

Models are represented using concrete syntax

- Visual
- Textual



No need for a 1-1 correspondence between abstract and concrete syntax elements



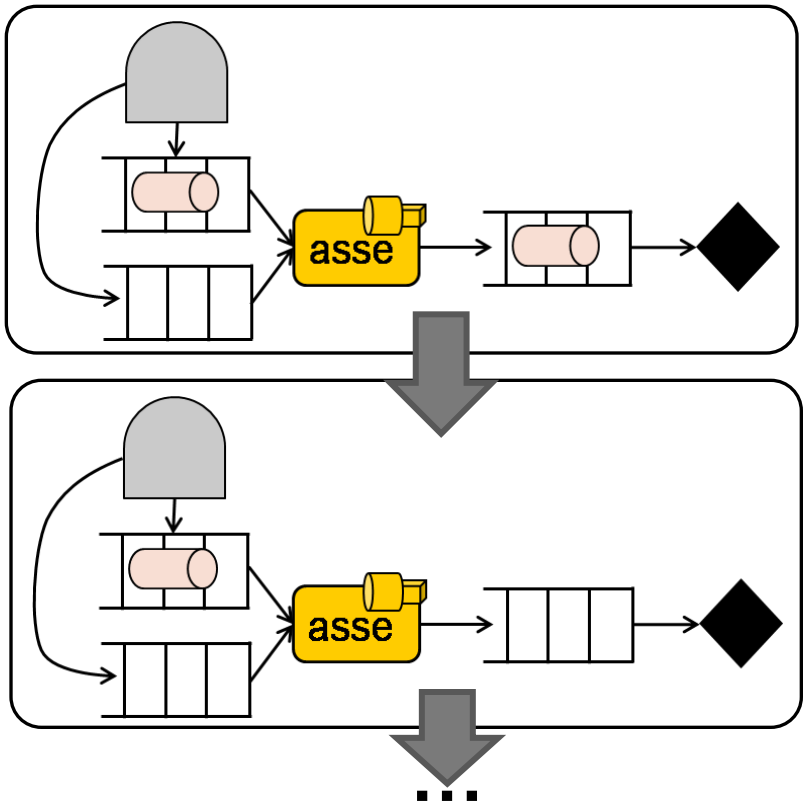
MODEL TRANSFORMATIONS

Models need to be manipulated for

- *Simulation*
- *Optimization/refactoring*

} ***in-place transformations***

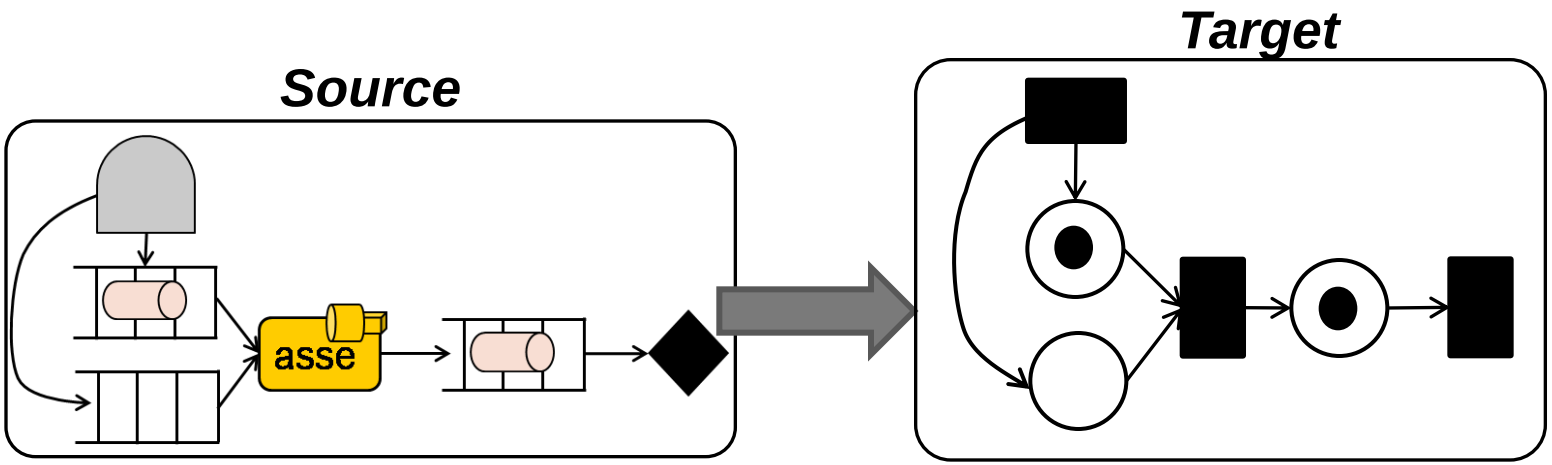
- Generating another model
- Generating code



MODEL TRANSFORMATIONS

Models need to be manipulated for

- Simulation
- Optimization/refactoring
- *Generating another model* } ***model-to-model transformations***
- Generating code

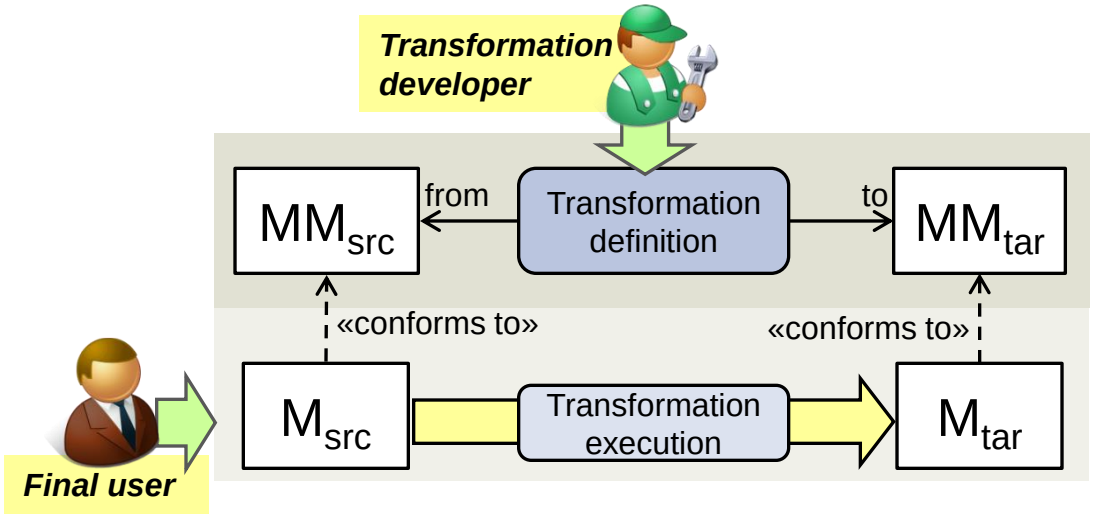


MODEL TRANSFORMATIONS

Models need to be manipulated for

- Simulation
- Optimization/refactoring
- Generating another model
- Generating code

} ***model-to-model transformations***

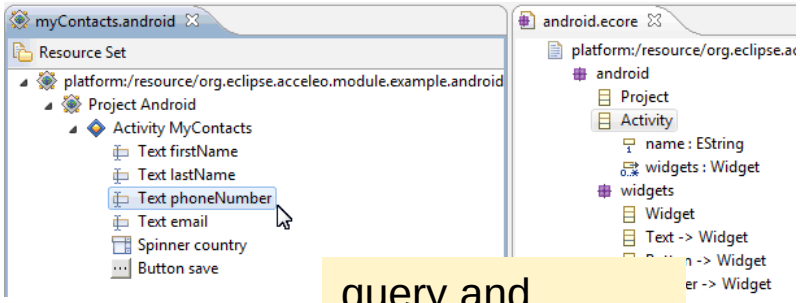


MODEL TRANSFORMATIONS

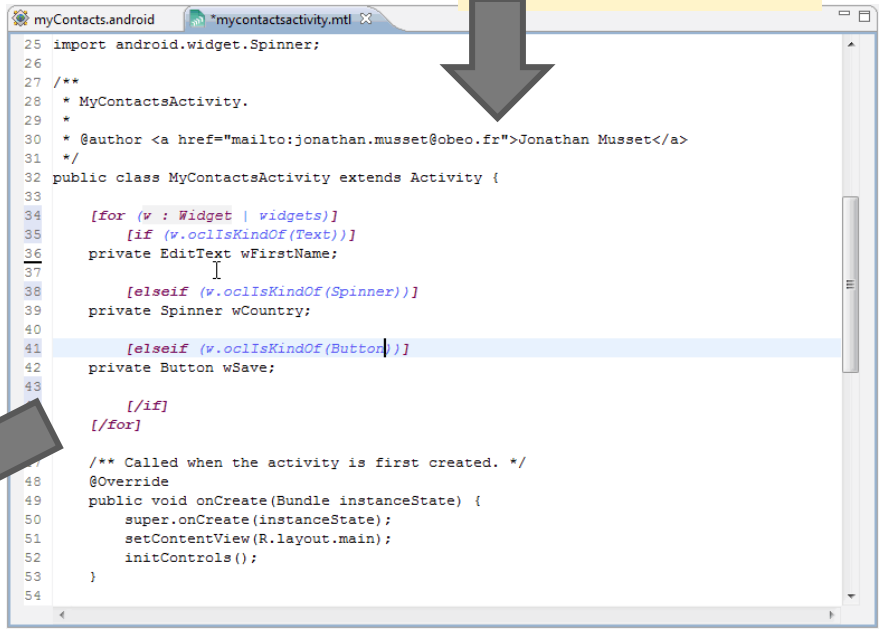
Models need to be manipulated for

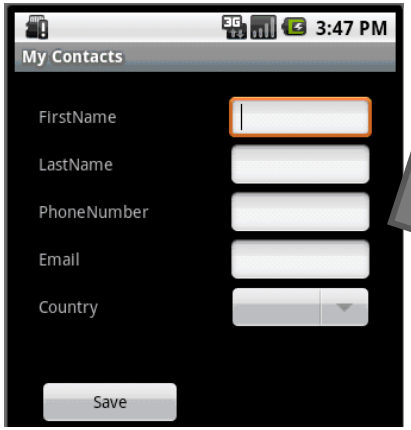
- Simulation
- Optimization/refactoring
- Generating another model
- *Generating code*

Template languages



query and model navigation





Very nice…
Where's the ugly?

SOME LIMITATIONS…

Model-driven Engineering

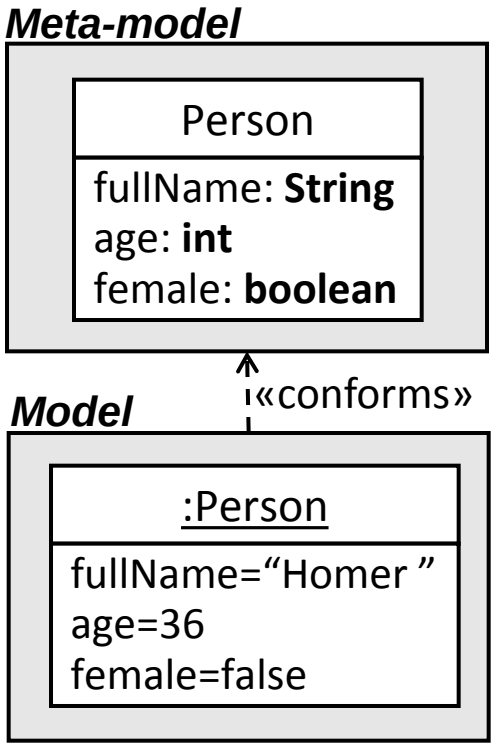
- Models are the principal artifacts
- Models conform to a meta-model

Objects are closed

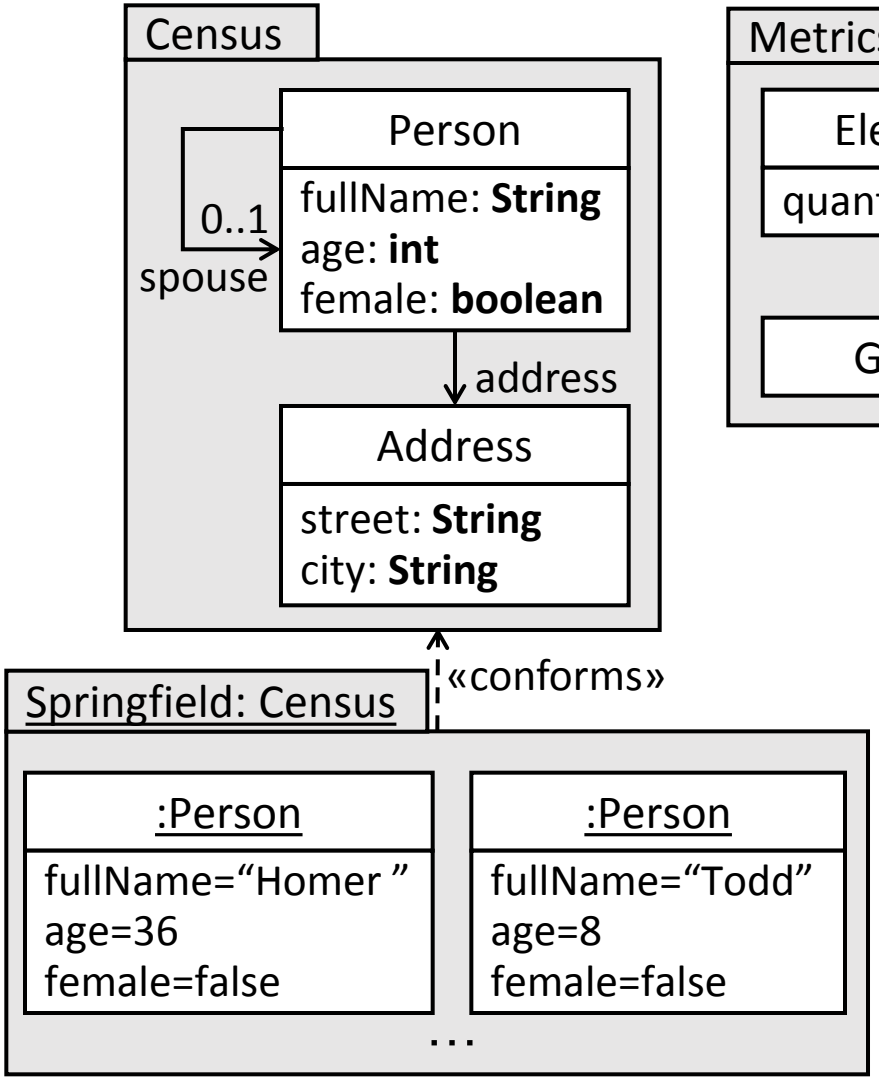
- Created using classes as templates
- Slots, types, and constraints cannot be changed

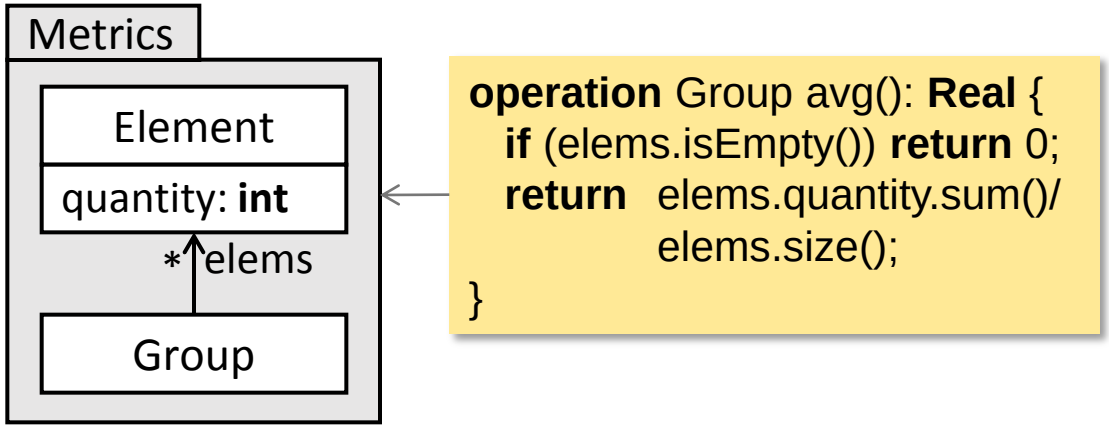
This rigidity makes some MDE scenarios difficult

- Reuse of model transformations
- Model extension
- Multi-view modelling
- …



MOTIVATION: REUSE



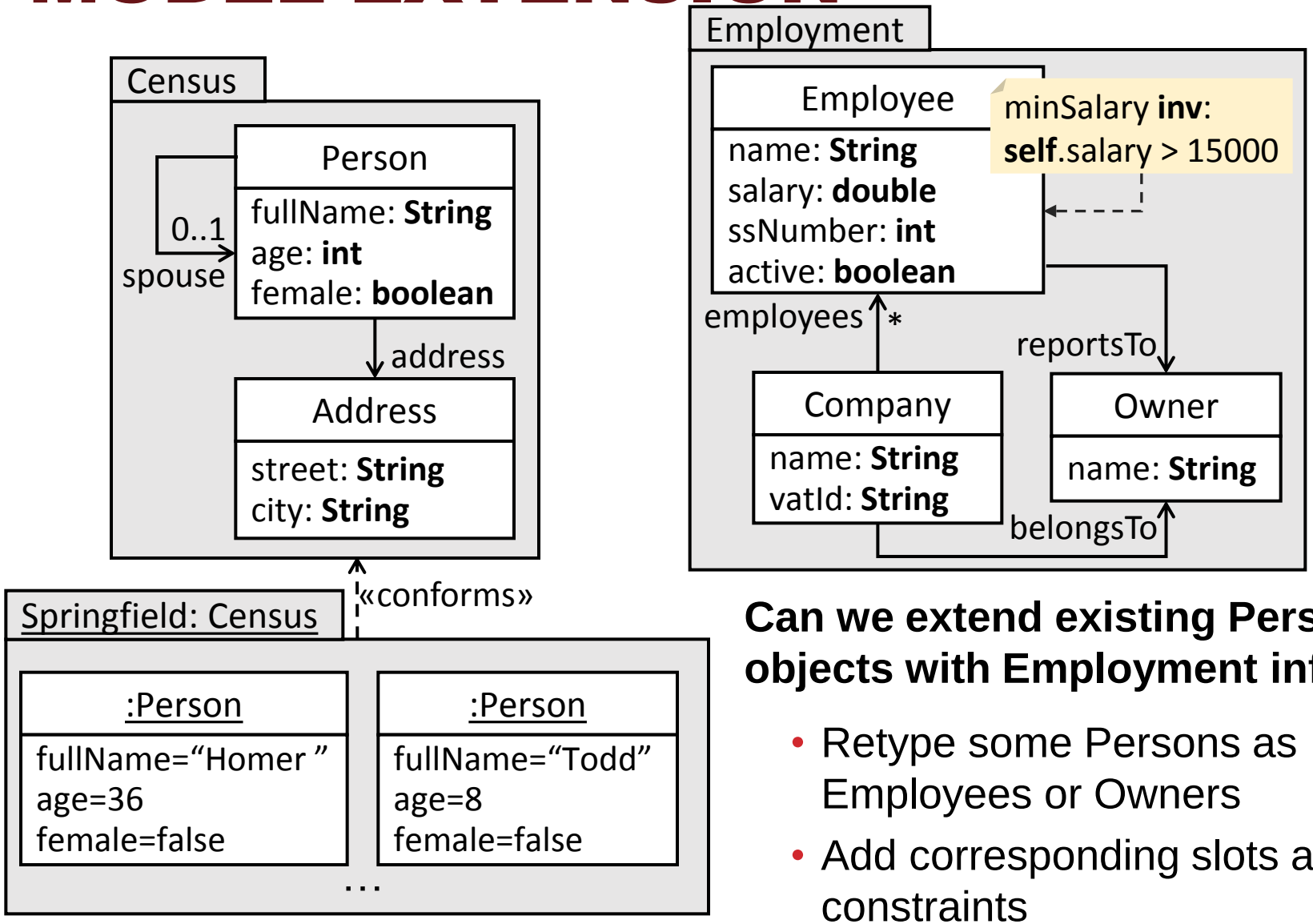


Calculate average age of all male adults in Springfield

Can we reuse operation `avg` on Census models?

- Retype some Persons as Elements
- Create a Group containing all male adults

MOTIVATION: MODEL EXTENSION



Can we extend existing Person objects with Employment info?

- Retype some Persons as Employees or Owners
- Add corresponding slots and constraints

IN THIS TALK...

New modelling mechanism: the *facet*

- brings flexibility and dynamism to modelling
- lightweight: facets are just objects

Objects become open

- can acquire and drop slots, types and constraints

Facet laws

- specify when objects acquire/drop facets

Practical implementation

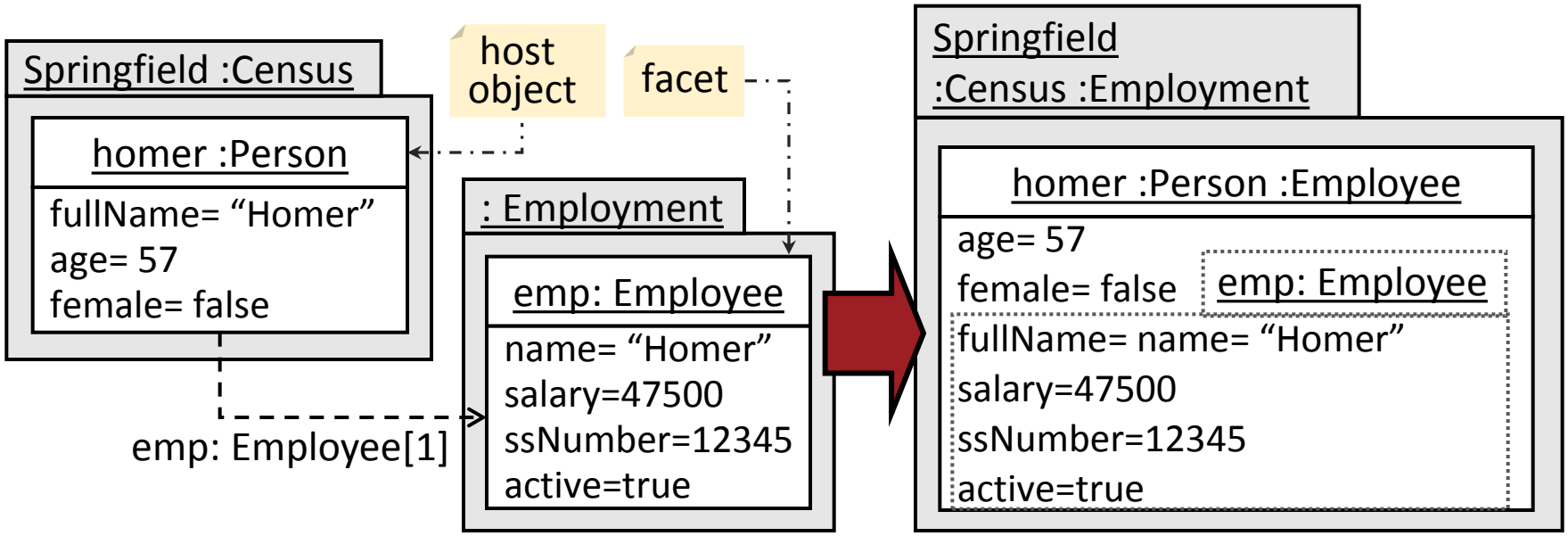
- on top of metaDepth, a textual meta-modelling tool

WHAT'S A FACET?

A facet is an object

- becomes part of another one(s), called the **host** object(s),
- the **slots** of the facet become **transparently accessible** from the host, which also acquires the **type** and **constraints** of the facet.

A host object can acquire and drop facets dynamically



WHAT'S A FACET?

Object `homer` **receives:**

- Slots (`name`, `salary`, `ssNumber`, `active`)
- Type (`Employee`)
- Constraints (`minSalary`)

From its `emp` **facet**

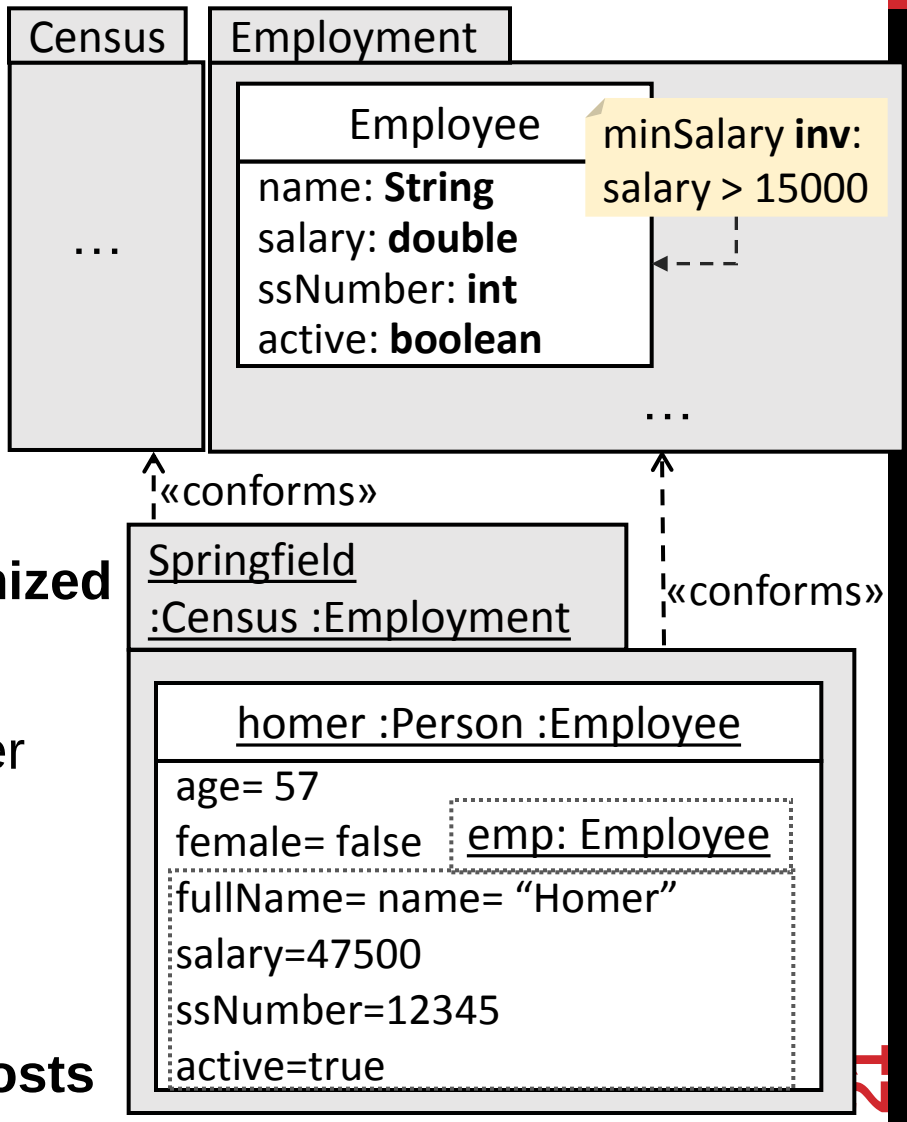
Host and facet slots may be synchronized

- `fullName` and `name`
- Changing either one modifies the other

Slot name ambiguity

- Resolved by facet name (`homer.emp`)

Shared facets and several facets in hosts



FACET MANAGEMENT

A DSL for adding/removing facets to objects

Selection of host objects by id

```
addFacet homer
  emp: Employment.Employee
  with {
      name = fullName [equality]
      salary = 22345
      ssNumber = 12345
      active = true
  }
```

Explicit host selection

Selection of host objects by properties

```
addFacet
  $Person.allInstances()→select(age>17)$
  emp: Employment.Employee with {…}
```

Query-based host selection

```
addFacet  ⟨h:Person, w:Person⟩ where
  $h.spouse=w$
  emp: Employment.Employee with {…}
```

Pattern-based host selection

(similar commands for removing a facet)

FACET MANAGEMENT

One-to-many and many-to-one host/facet relations are supported

```
addFacet homer
  dayJob: Employment.Employee with {
    name      = fullName              [equality]
    salary    = 15000
    ssNumber  = 12345
    active    = true
  }
  nightJob: Employment.Employee with {
    name      = fullName              [equality]
    salary    = 16400
    ssNumber  = dayJob.ssNumber       [equality]
    active    = dayJob.active         [equality]
  }
```

Several facets in same host object

```
addFacet
$Person.all()→select(age>17)$
  emp: Employment.Employee
  with {…} reuse
```

Facet shared among all selected host objects

REACTIVE FIELD ADAPTERS

```
addFacet homer
  emp: Employment.Employee
  with {
    ssNumber = 12345                  // value semantics: literals
    salary   = $100*self.age$         // value semantics: expressions
    name     = fullName [equality]    // reference semantics: bx synchronization
    active   = [self.age < 65]        // reference semantics: reactive field adapter
    }
```

Coupled change dependencies

- active = [**self**.age < 65] → When age changes, active is updated
- name = fullName [**equality**] → Eq. to name = [fullName] [fullName=name]

N-ary depedencies

- salary = [100*__self__.age] [rich = __self__.salary > 10000]
- ill-behaved: name = [fullName] [fullName = ’Mr. ’.concat(name)]
- Safety policy: each field is evaluated once within a cycle

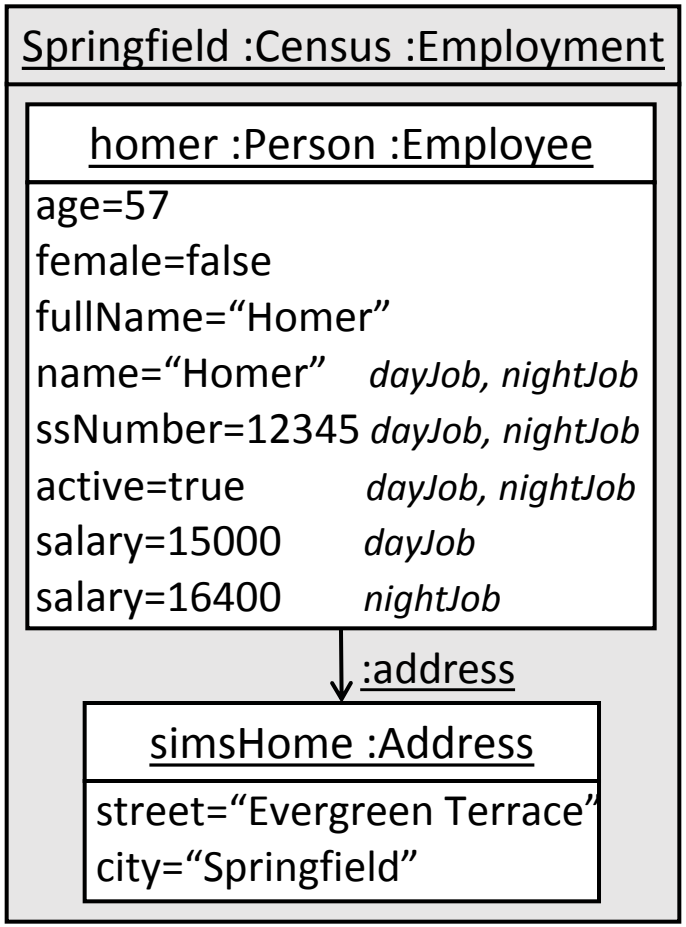
MODEL SCENES

Different visualizations for a model

- "Scenes"

Total scene

- Default visualization



Total scene

MODEL SCENES

Different visualizations for a model

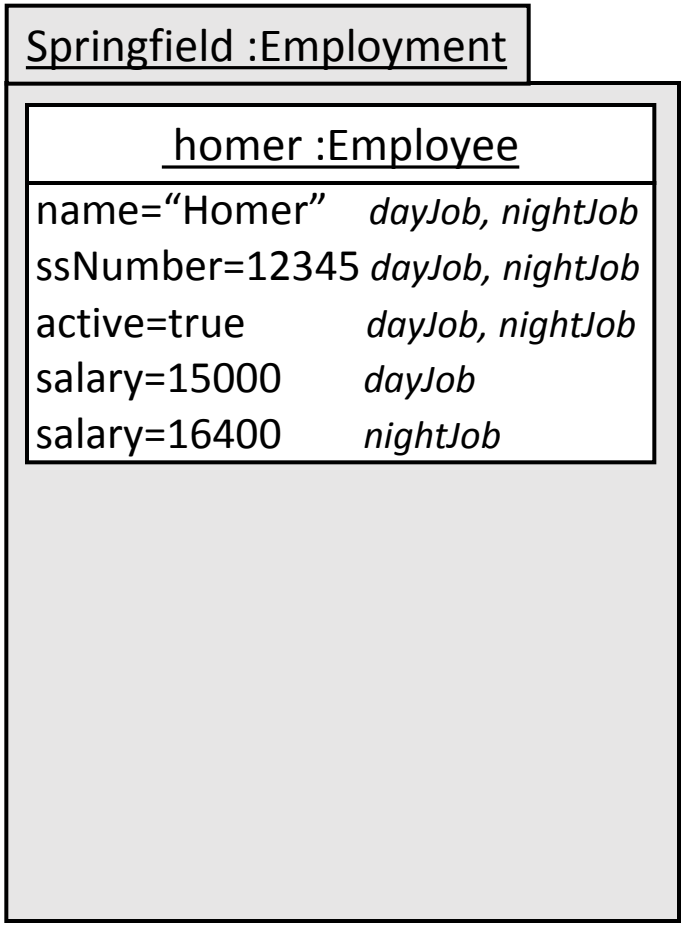
- "Scenes"

Total scene

- Default visualization

Sliced scene

- W.r.t. a given facet meta-model



Scene sliced by Employment

MODEL SCENES

Different visualizations for a model

- “Scenes”

Total scene

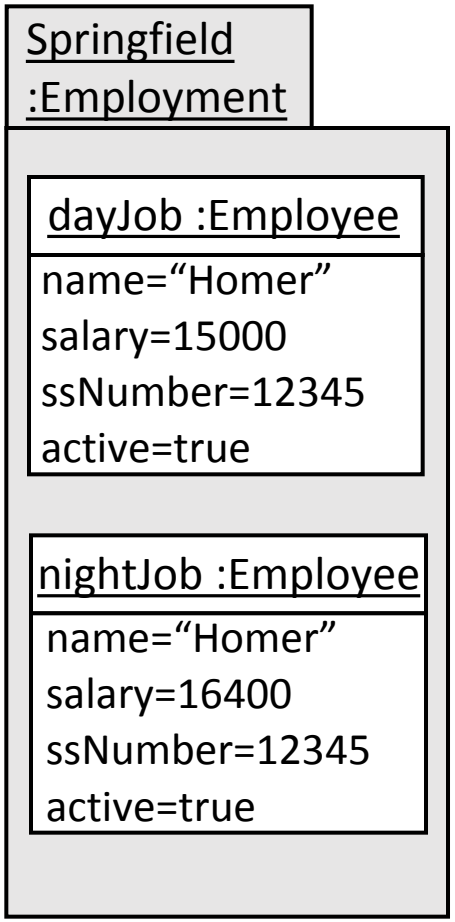
- Default visualization

Sliced scene

- W.r.t. a given facet meta-model

Granulated

- Shows facets typed w.r.t. a certain meta-model



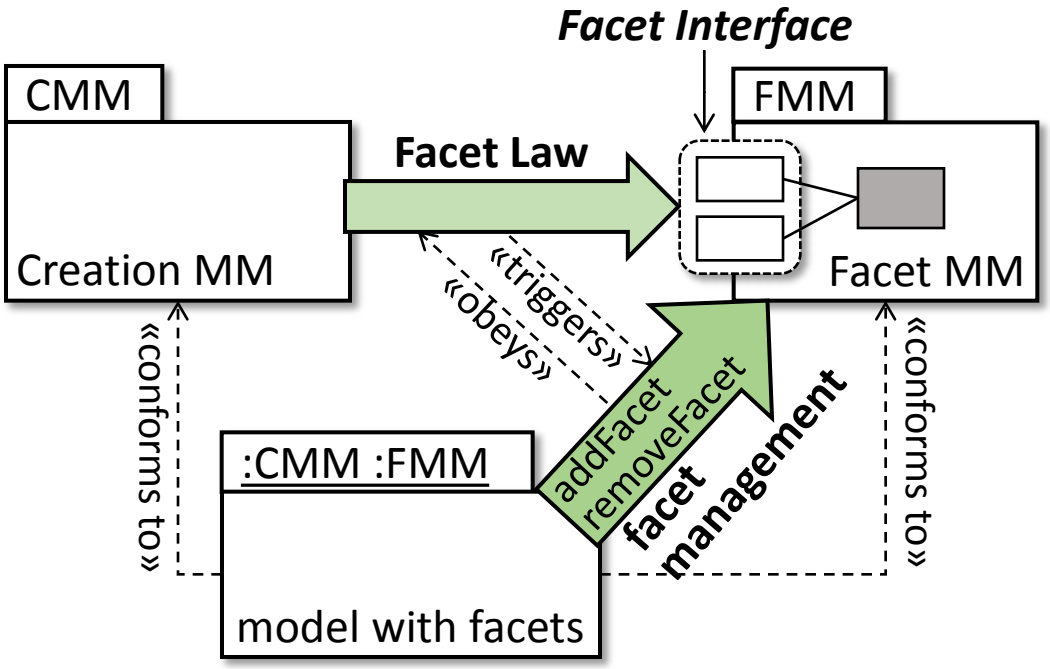
Scene granulated by Employment

FACET LAWS AND INTERFACES

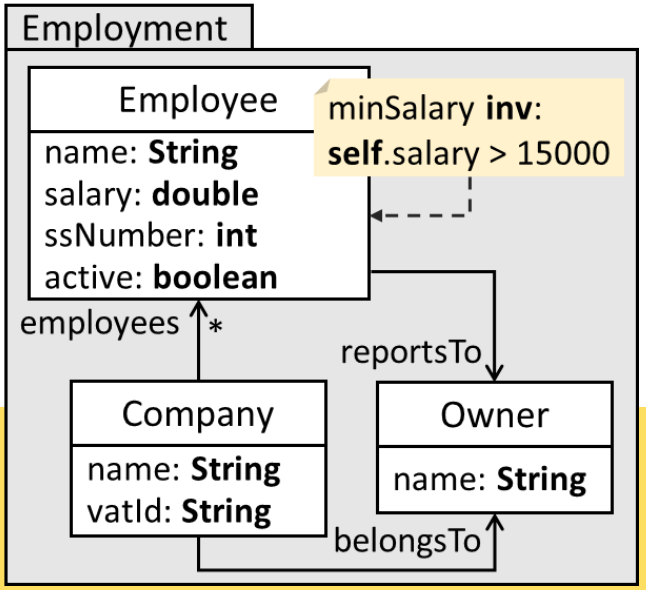
Opportunistic vs planned handling of facets

Control which elements can be used as facets

Declarative specification of conditions for acquiring/droping facets



FACET INTERFACES



FacetInterface for Employment {
public: **all**
compatible: [Employee, Owner]
constraints: Employee.repToIrreflexive= $**self**.reportsTo.excludes(**self**)$
}

Restricts how a meta-model can be used for facet-based modelling

Declares

- classes that can be used to create facets
- allowed combinations
- extra well-formedness constraints (eg., due to facet combination)

FACET LAWS

Specs stating when host objects should acquire/drop a facet

- must/may

Can add additional constraints and set default values

```
FacetLaws for Census with Employment {
    must extend <p:Person> where $p.age>17$
        with work:Employee with {
            name = fullName [equality]
            salary = 24000
            minLocalSalary: $ self.salary>16000 $
            retirement: $ self.age>65 implies not self.active $
        }
}
```

Setting `homer.age:=16` **makes homer drop the** `work` **facet**

FACET LAWS

Check manually issued addFacet/removeFacet commands

- Should conform to the facet laws, if defined

Check faceted models for consistency

- Models should satisfy the laws

To complete addFacet commands

- Take default values and slot relations from the laws

To constraint facets

- By adding extra constraints

To automate facet acquisition/loss

- Via the “must” extension laws
- addFacet/removeFacet automatically issued

TOOL SUPPORT

MetaDepth (http://metaDepth.org)

- Textual multi-level modelling tool
- Epsilon languages for model management (EOL, ETL, EGL)

Facets, facet handling, interfaces, laws

Mirror fields

Triggered constraints (add/drop facets)

Census meta-model

```
Model Census {
  Node Person{
    name: String;
    age: int;
    female: boolean = true;
    spouse: Person[0..1];
    address: Address[1];
  }
  Node Address {
    street: String;
    city: String;
  }
}
```

EOL program

```
var p : Person := new Person;
p.age    := 23;    // implicitly creates an Employee facet (as p.age > 17)
p.salary := 15100; // OK, as p has now an Employee facet
p.age    := 16;    // p loses its Employee facet (as p.age <= 17)
p.salary := 21000; // Error! p has no Employee facet
```

EVALUATION

Based on five scenarios

- Integration of annotation models
- Reuse of model management operations
- Multi-view modelling
- Multi-level modelling
- Language product lines

Comparison with solutions using alternative techniques

- Cross-referencing, EMF profiles, a-posterioti typing
- Model adaptation, a-posteriori typing, concepts, model typing
- Central repository, OpenFlexo, OSM, Vitruvius
- MetaDepth, Melanee, MultEcore, ML2
- Model templates, DeltaEcore, VML*, SmartAdapters, etc

INTEGRATING ANNOTATION MODELS

Annotation models widely used in MDE:

- Concrete syntax (CS) representation, uncertainty, variability, access control, etc.

Graphical CS support

- Meta-model
- Simple visualizer
- Facet laws to assign CS to domain meta-models

We obtain for free:

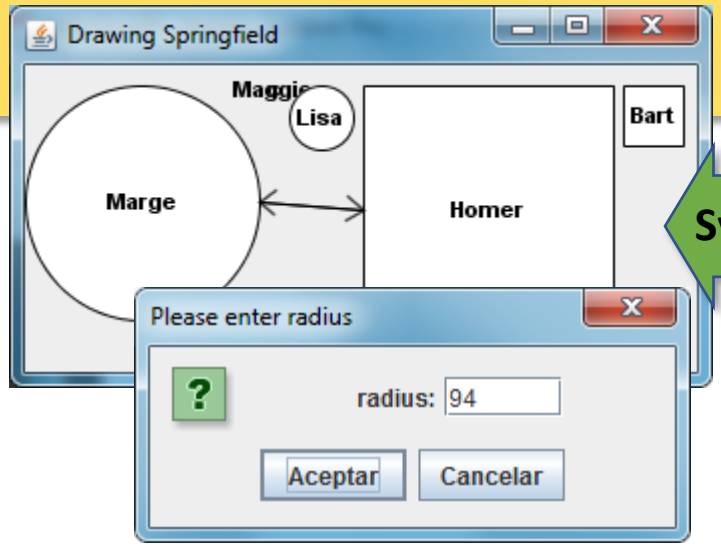
- Bidirectional synchronization textual/graphical CS.

CS meta-model

```
Model ConcreteSyntax {
  abstract Node GraphicalElem {
    x, y       : int;
    label      : String;
    linkedTo   : GraphicalElem[*];
  }
  Node Rectangle : GraphicalElem {
    width, height : int;
  }
  Node Circle : GraphicalElem {
    radius : int;
  }
}
```

CS FOR CENSUS

```
FacetLaws for Census with ConcreteSyntax {
 must extend <p:Person> where $p.female$ with c:Circle with {
    label      = name    [equality]
    linkedTo = spouse  [equality]
    radius     = [2*age] [age = radius/2]
 }
 must extend <p:Person> where $not p.female$ with r:Rectangle with {
    label      = name    [equality]
    linkedTo = spouse  [equality]
    height     = age     [equality]
    width      = age     [equality]
 }
}
```



Synch.

```
Census Springfield {
  Person marge {
    fullName = "Marge";
    age      = 47;
    spouse   = homer;
    address  = simsHouse;
  }
  …
}
```

(scene sliced by Census)

CAN WE DO THIS DIFFERENTY?

Approach	Dynamic	Sharing	Field access	Typing	Bx change
Cross-ref. (base.)	no	yes	navigation	no	no
EMF profiles [1]	limited	yes	navigation	no	no
A-posteriori [2]	limited	limited	transparent	yes	limited
Facets	yes	yes	transparent	yes	yes

Cannot be used to fully solve this case study:

Cross-referencing, EMF profiles

- No direct support for conditional styles (different CS based on female)
- No direct support for bidirectional changes

A-posteriori typing

- Cannot map slots like x and y

[1] Langer, Wieland, Wimmer,Cabot. *EMF profiles: a lightweight extension approach for EMF models*. JOT 11,1(2012),1–29.

[2] de Lara, Guerra. *A posteriori typing for model-driven engineering: Concepts, analysis, and applications*. ACM TOSEM 25, 4(2017),31:1–31:60.

(DYNAMIC) PRODUCT LINES

"*SE methods for creating a collection of similar software systems from a shared set of software assets using a common means of production*"

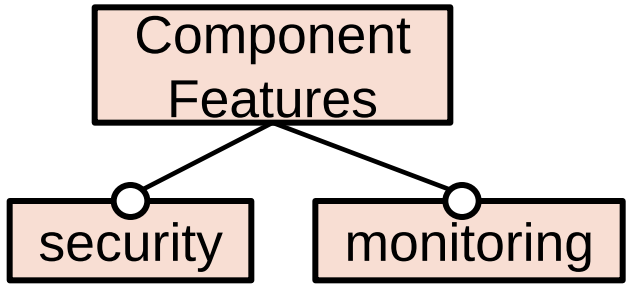
(Dynamic) Language product lines

```
Model Components {
 abstract Node NamedElement {
    name : String;
 }
 Node Component : NamedElement {
    ports : Port[1..*];
 }
 abstract Node Port : NamedElement ;
 Node InputPort : Port;
 Node OutputPort : Port {
    target : InputPort[1..*];
 }
}
```

Base language definition

```
// variability model
Model ComponentFeatures {
   Node FeaturedElement {
     security : boolean = false;
     monitoring : boolean = true;
   }
}
```

Feature model



(DYNAMIC) PRODUCT LINES

```
Model ComponentFacets { // facet metamodel
  Node Cipher { // to be added to ports when security is selected
    blockSize : int;
    key : String;
    nRounds : int;
  }
  Node Monitor { // to be added to components when monitoring is selected
     activeRate : double = 0.0;
     powerConsumption : double = 0.0;
   }
}
```

Meta-model fragments to be added when the configuration changes

(DYNAMIC) PRODUCT LINES

All elements share a configuration

```
FacetLaws for Components with ComponentFeatures {
  must extend <n:NamedElement> with cfg: FeaturedElement with {
   security = false
   monitoring = true
  } reuse
}
```

Facets are added depending on the chosen configuration

```
FacetLaws for Components, ComponentFeatures with ComponentFacets {
  must extend <p:Port & FeaturedElement> where $p.security$
    with c : Cipher with {
      blockSize = 32
      key = "915F4619BE41B2516355A50110A9CE91"
      nRounds = 12
    }
   must extend <c:Component & FeaturedElement> where $c.monitoring$
    with m: Monitor with {
      …
    } reuse
}
```

RELATED WORKS

Role-based modelling (eg., Lodwick [1], CROM [2])

- Facets fulfill most typical features of role-based languages
- Heavyweight (role, natural, compartment types)
- Practical integration with MDE: inheritance, attribute/slot handling, OCL constraints, integration with model management languages

[1] Steimann. On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng. 35,1(2000),83–106.

[2] Kühn, Böhme, Götz,Aßmann. A combined formal model for relational context-dependent roles. In SLE'15. pp.:113–124.

A-posteriori typing

- Can dynamically add/remove types
- Cannot add slots or constraints

de Lara, Guerra. *A posteriori typing for model-driven engineering: Concepts, analysis, and applications*. ACM TOSEM 25, 4(2017),31:1–31:60.

CONCLUSIONS AND FUTURE WORK

Facets add flexibility and dynamicity to modelling

- make objects open
- acquire/drop slots, types and constraints dynamically
- reactive synchronization of fields

Facet interfaces and laws

- property-based facet acquisition and drop

Some scenarios where facets present advantages

Implementation on top of metaDepth

Future work

- Improve tooling
- Interaction with behaviour specifications
- Combine operations defined in host objects and facet
- Static analysis

THANKS!

http://miso.es
Juan.deLara@uam.es
@miso_uam

http://metadepth.org