A CSP Approach for Meta Model Instantiation

Adel Ferdjoukh, Anne-Elisabeth Baert, Annie Chateau, Rémi Coletta and Clémentine Nebut

Montpellier, France

Summary

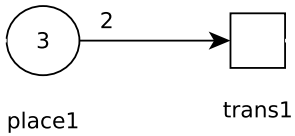
Summary

Model Driven Engineering (MDE)

- A recent paradigm recommending the intensive use of structured models [?].
- A model is defined by a meta model.

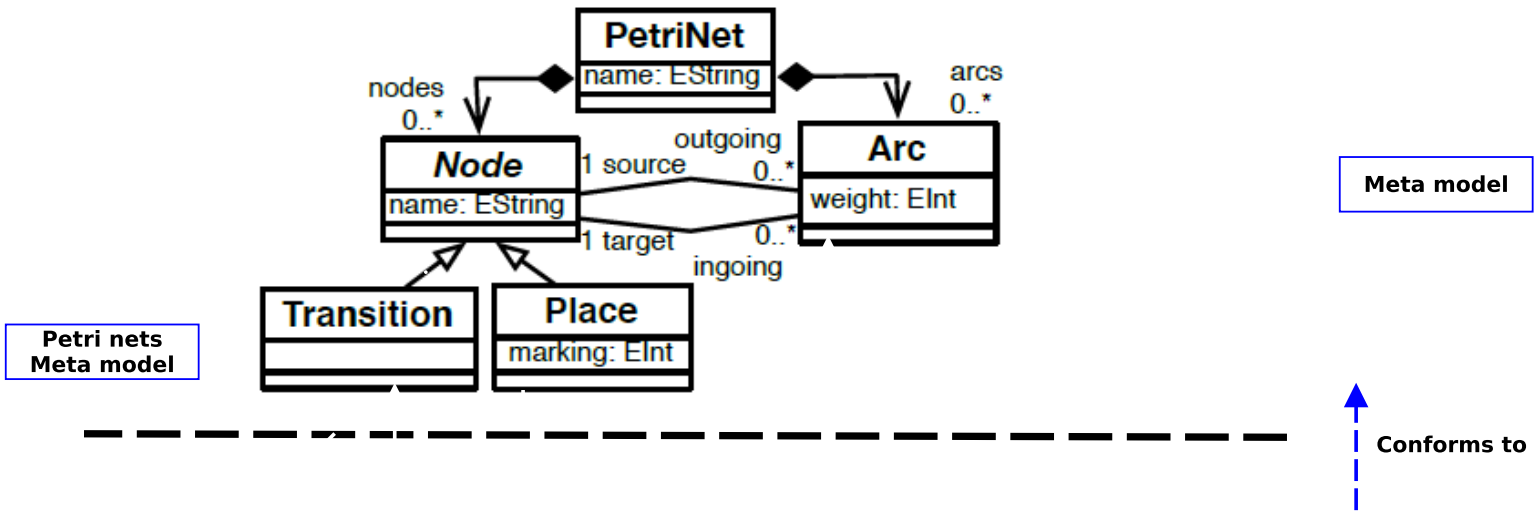
Models,meta models and OCL constraints

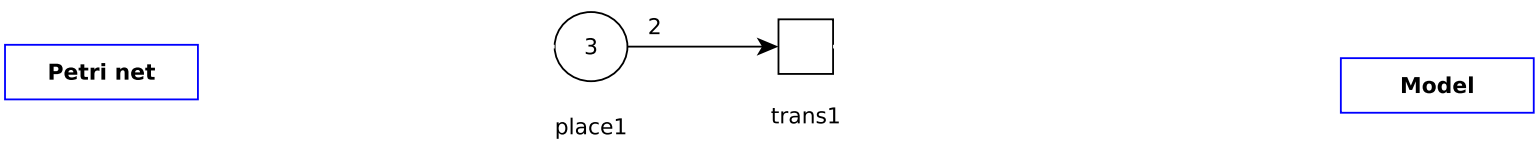
Petri net



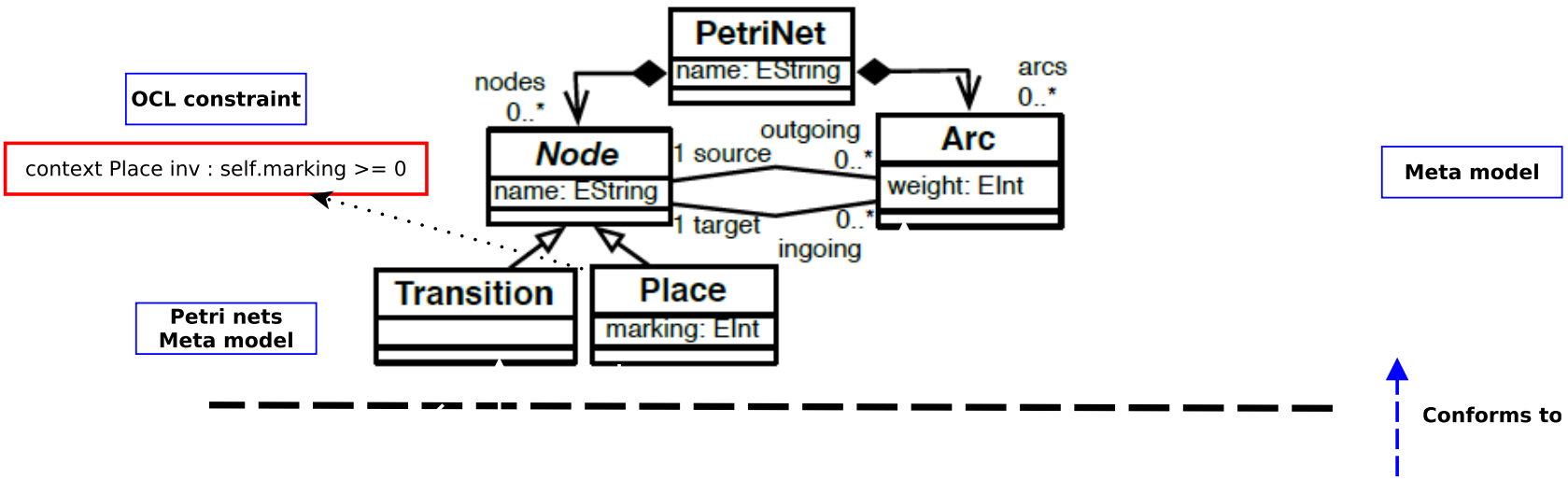
Model

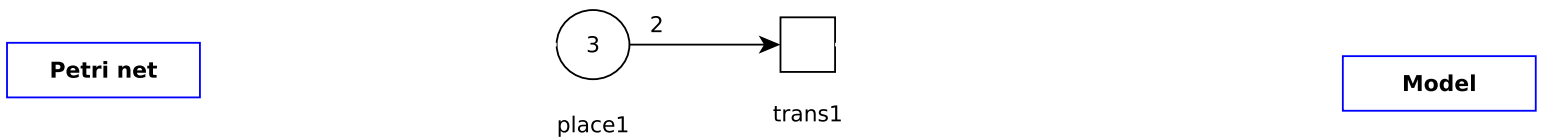
Models,meta models and OCL constraints



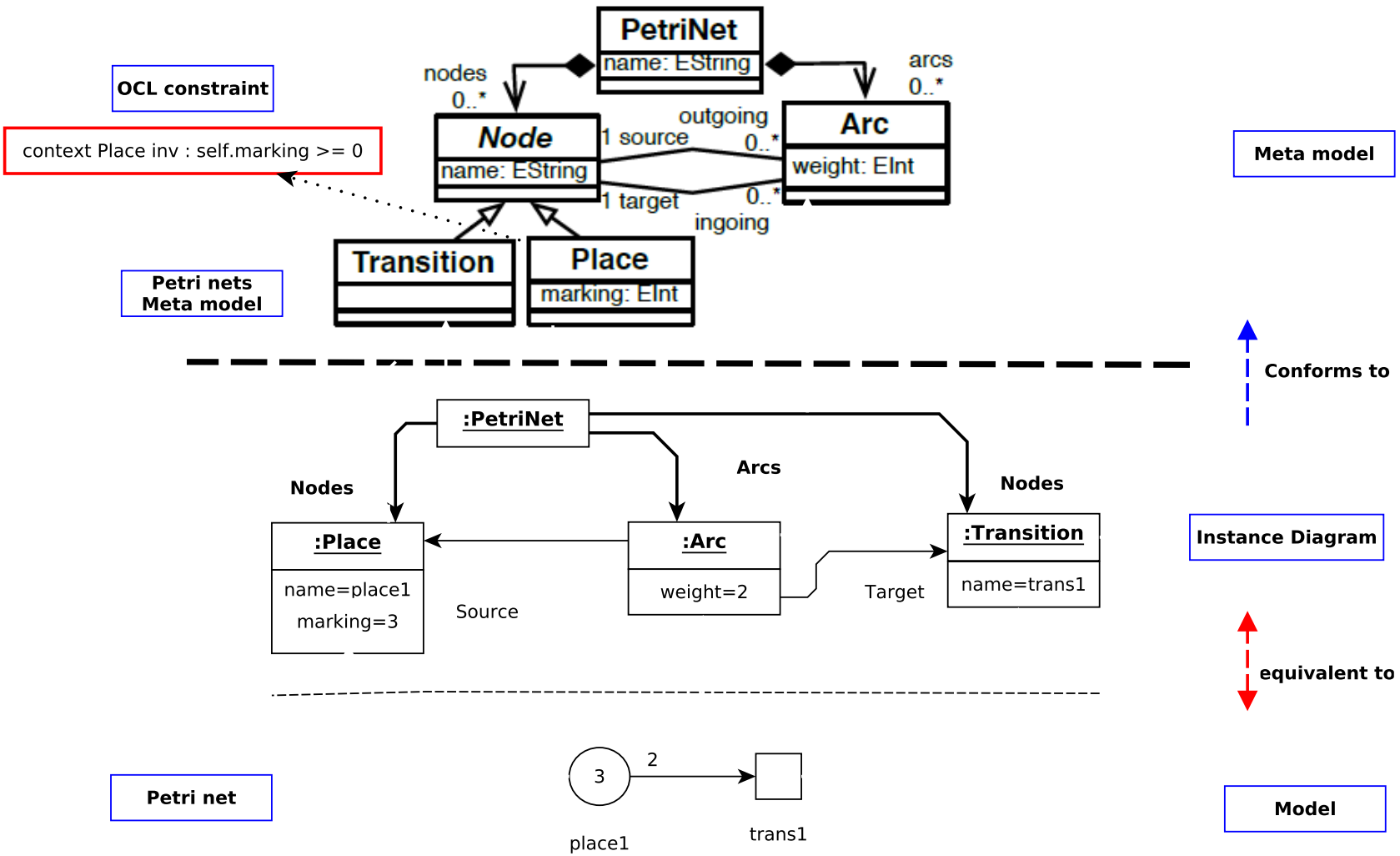


Models,meta models and OCL constraints

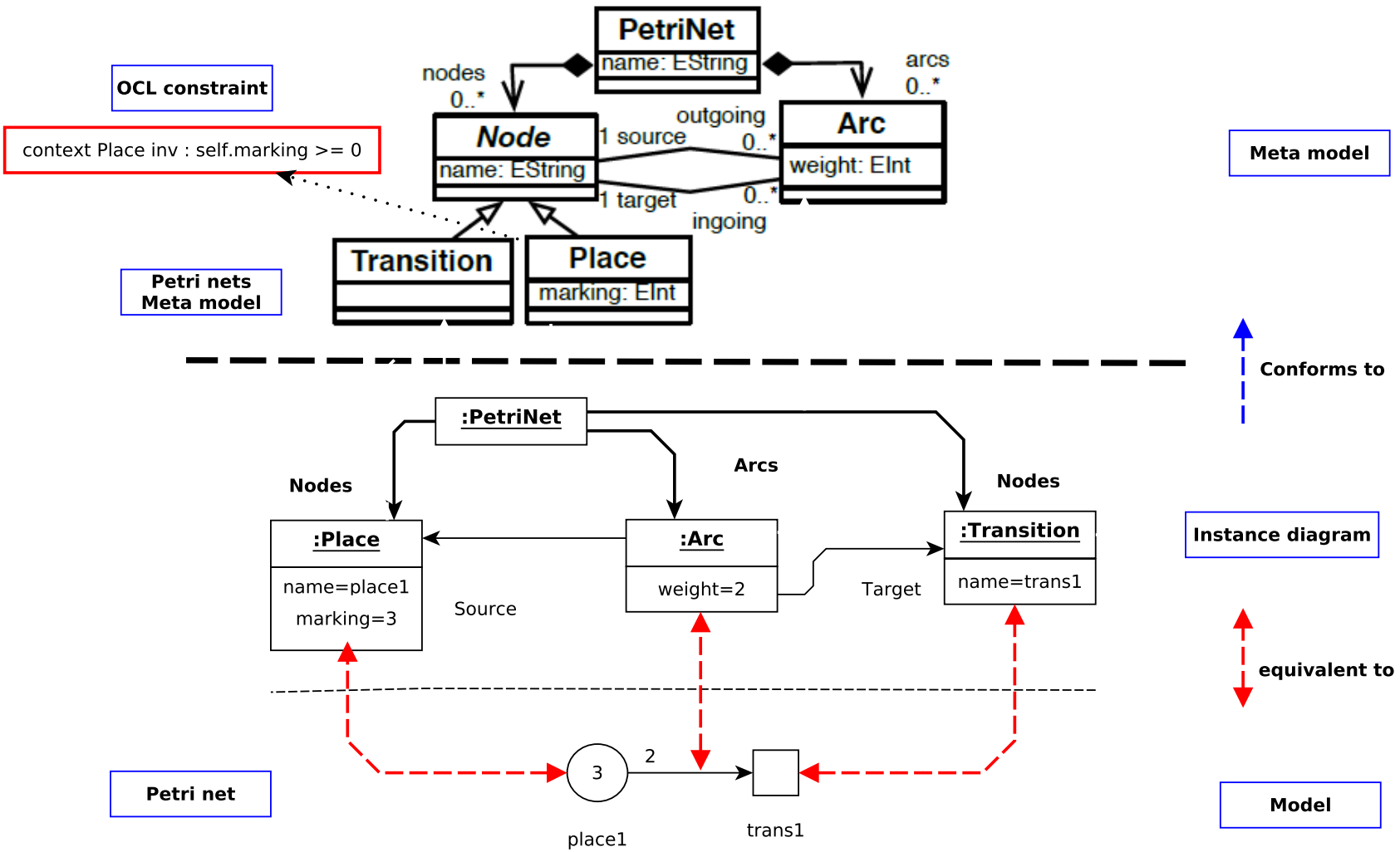




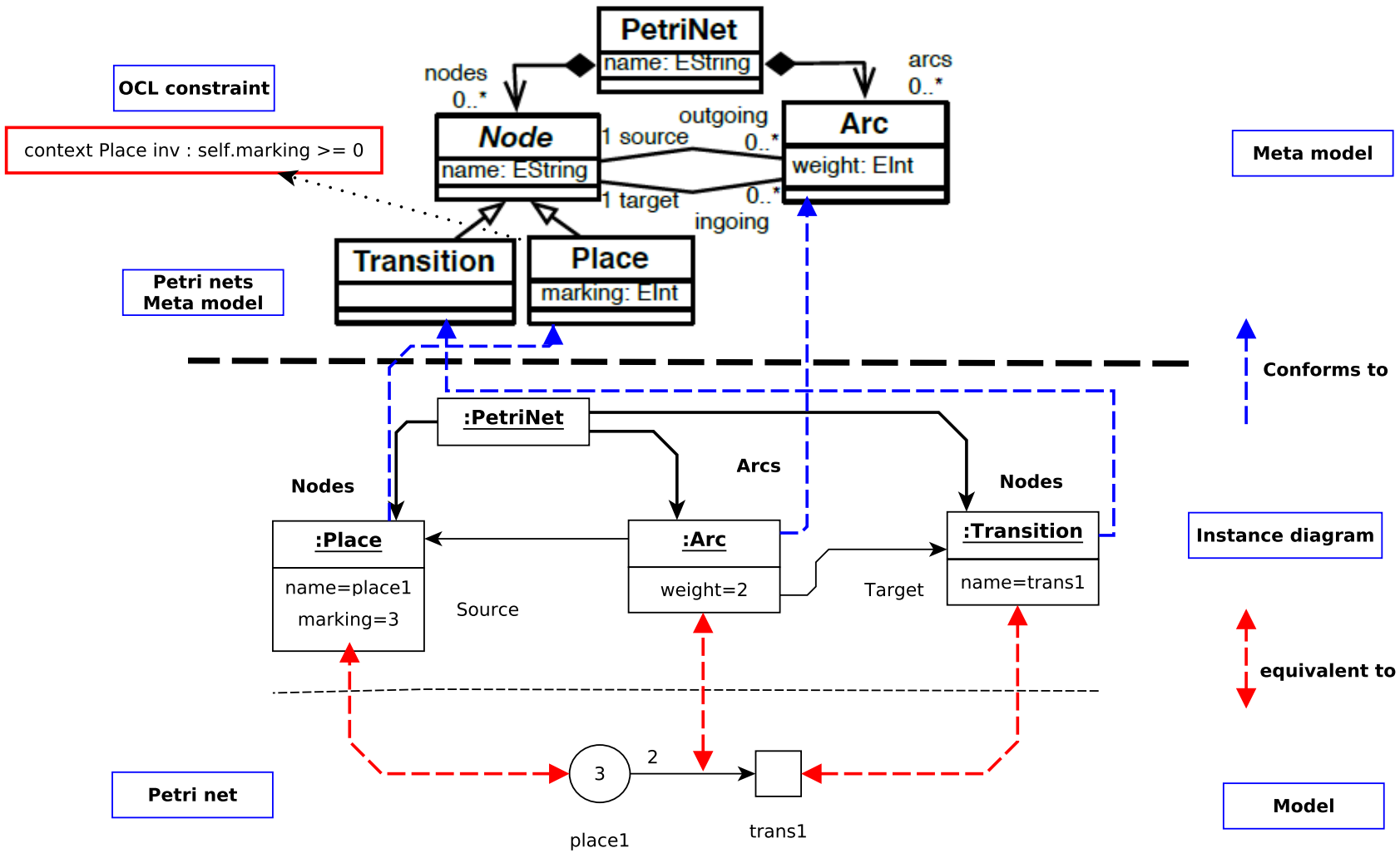
Models,meta models and OCL constraints



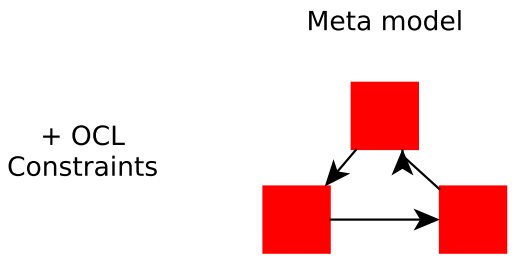
Models,meta models and OCL constraints



Models,meta models and OCL constraints



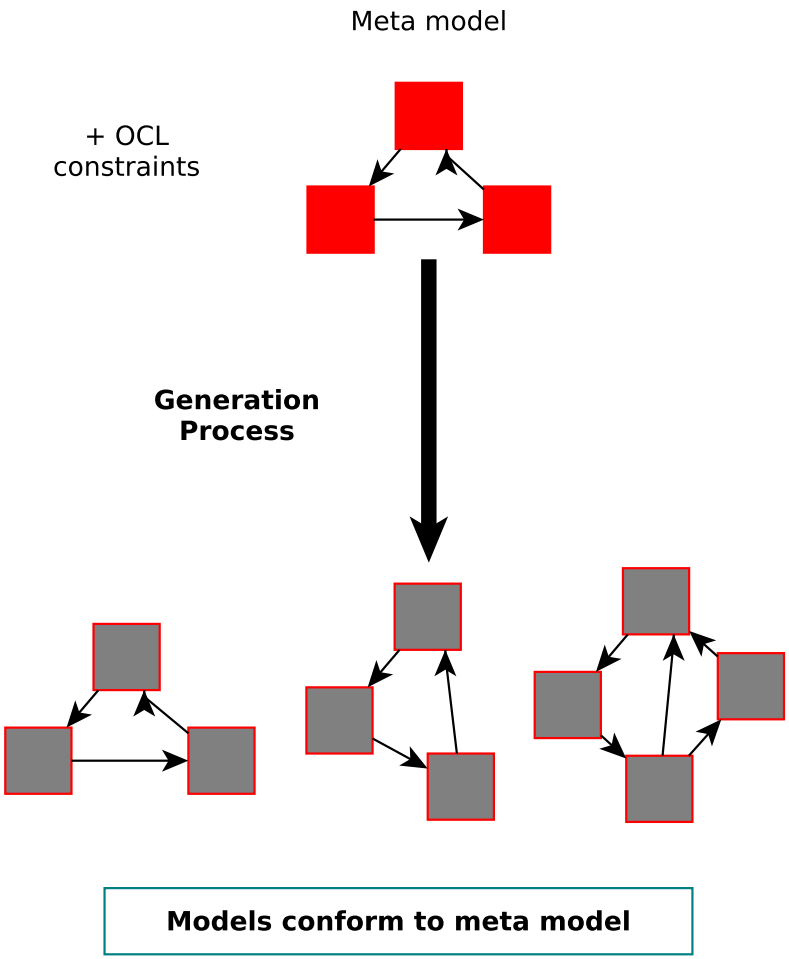
Model Generation



Why generating models ?

- Automatic validation of meta models.
- Obtain test data for model transformations.

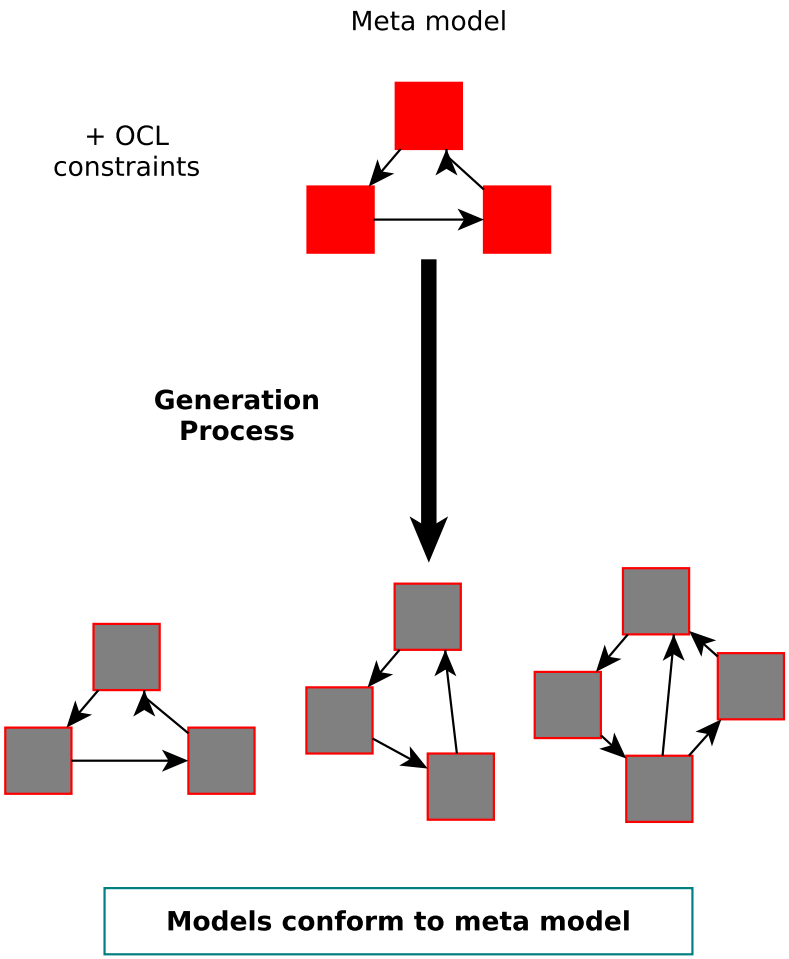
Model Generation



Why generating models ?

- Automatic validation of meta models.
- Obtain test data for model transformations.

Model Generation



Why generating models ?

- Automatic validation of meta models.
- Obtain test data for model transformations.

Model Generation

Generation process expected properties

- **Scalability**: Generation process must scale to large models and meta models.
- Validity.
- Flexibility.
- Diversity.

Model Generation

Generation process expected properties

- Scalability.
- **Validity**: Models must conform to a given meta model and respect OCL constraints.
- Flexibility.
- Diversity.

Model Generation

Generation process expected properties

- Scalability.
- Validity.
- **Flexibility**: It should be easy to parameterize the generation process in order to add other constraints.
- Diversity.

Model Generation

Generation process expected properties

- Scalability.
- Validity.
- Flexibility.
- **Diversity**: Models should be as representative as possible of the meta model.

Using CSP to Generate Models

Existing CSP approach

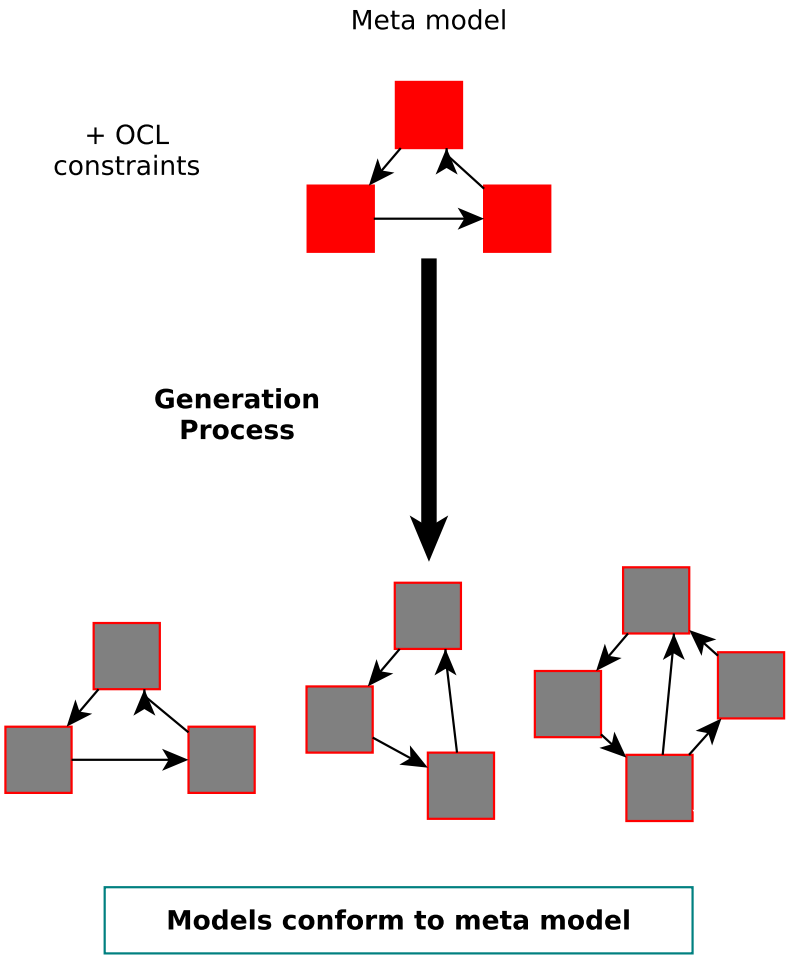
- *Cabot's et al.* CSP approach [?, ?].

So why do we choose a CSP approach ?

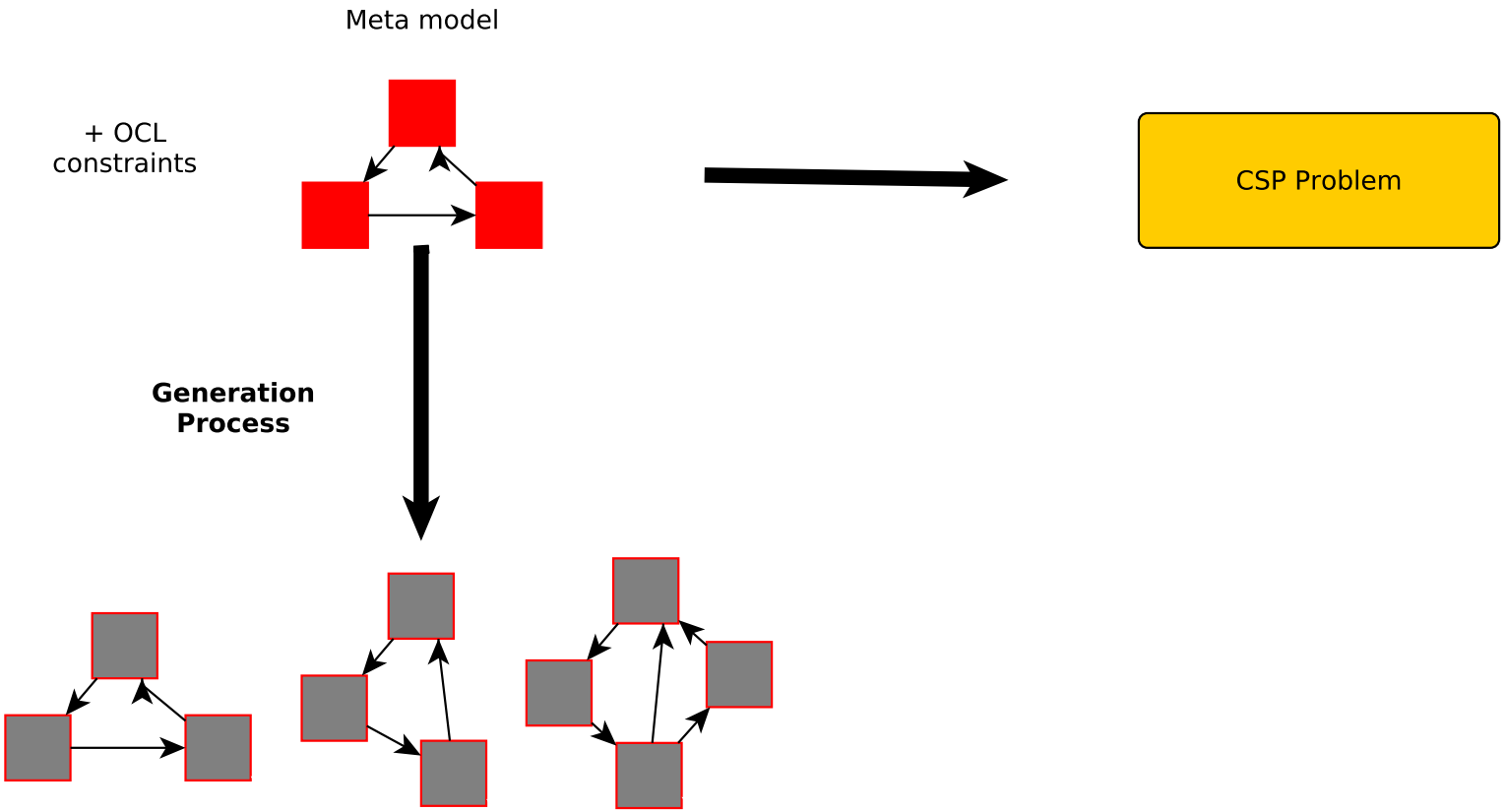
- Lack of flexibility of other approaches.
- CSP allows taking into account OCL constraints.
- Existing CSP approach does not scale.

Summary

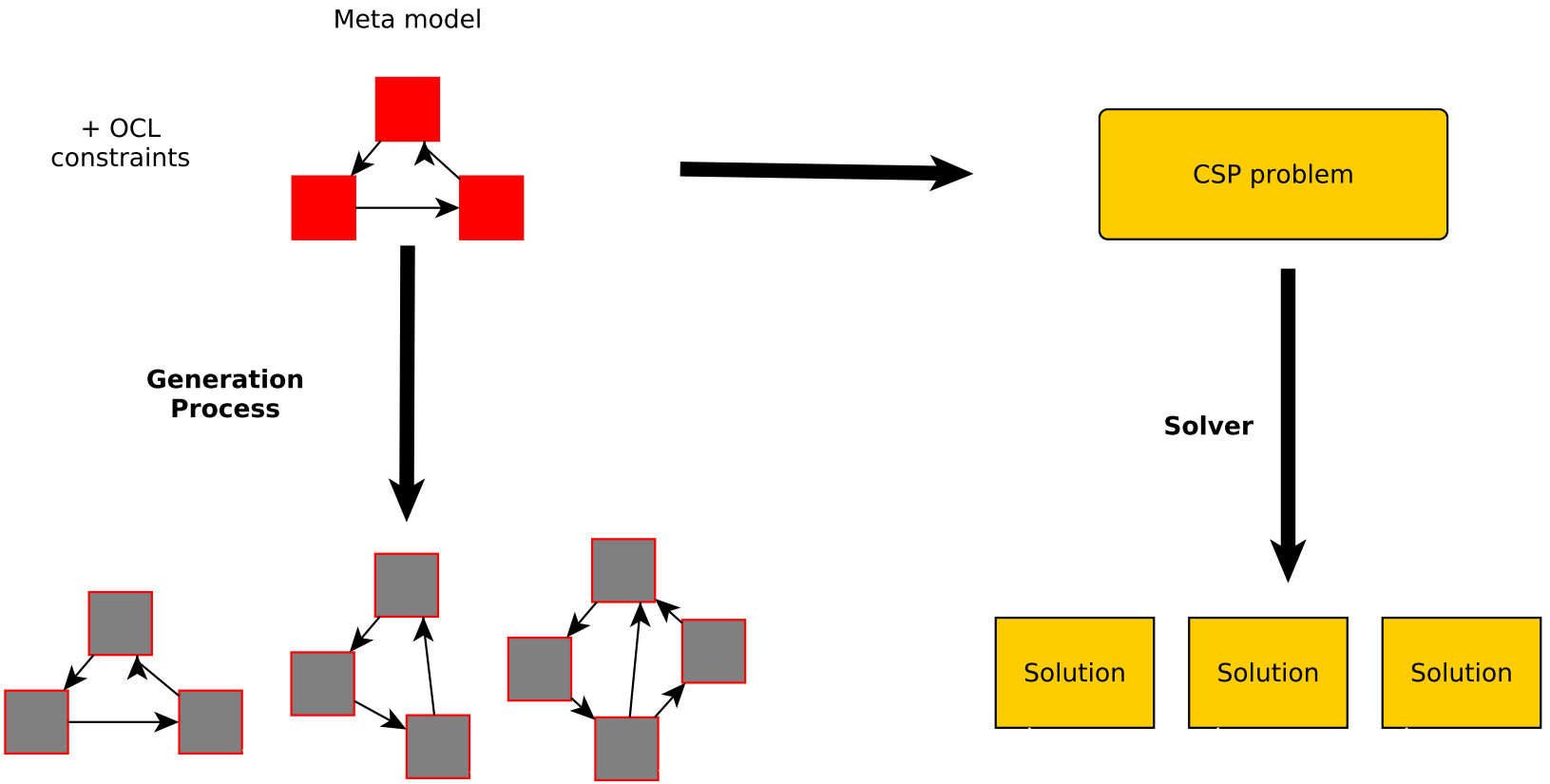
CSP for Model Generation



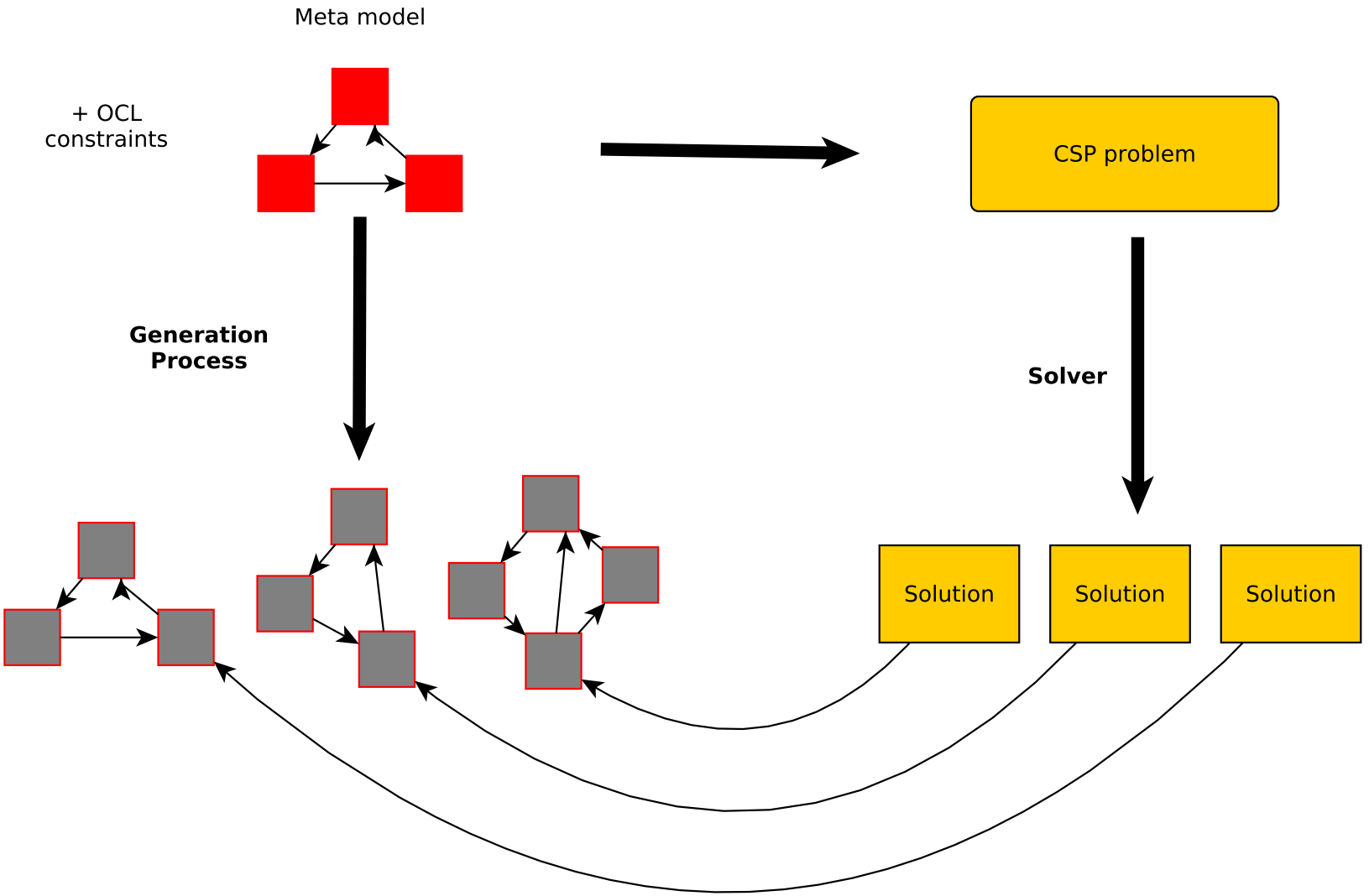
CSP for Model Generation



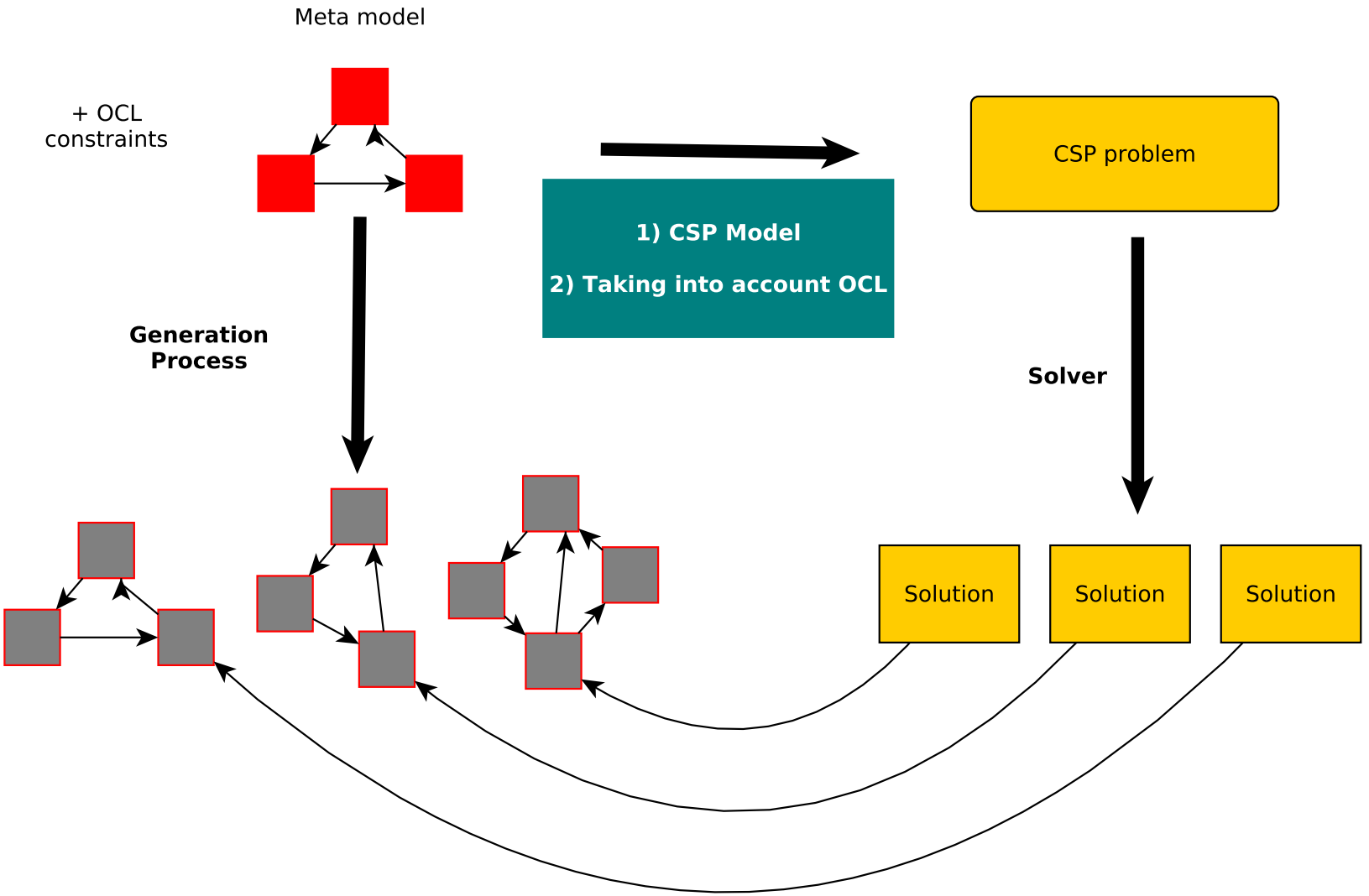
CSP for Model Generation



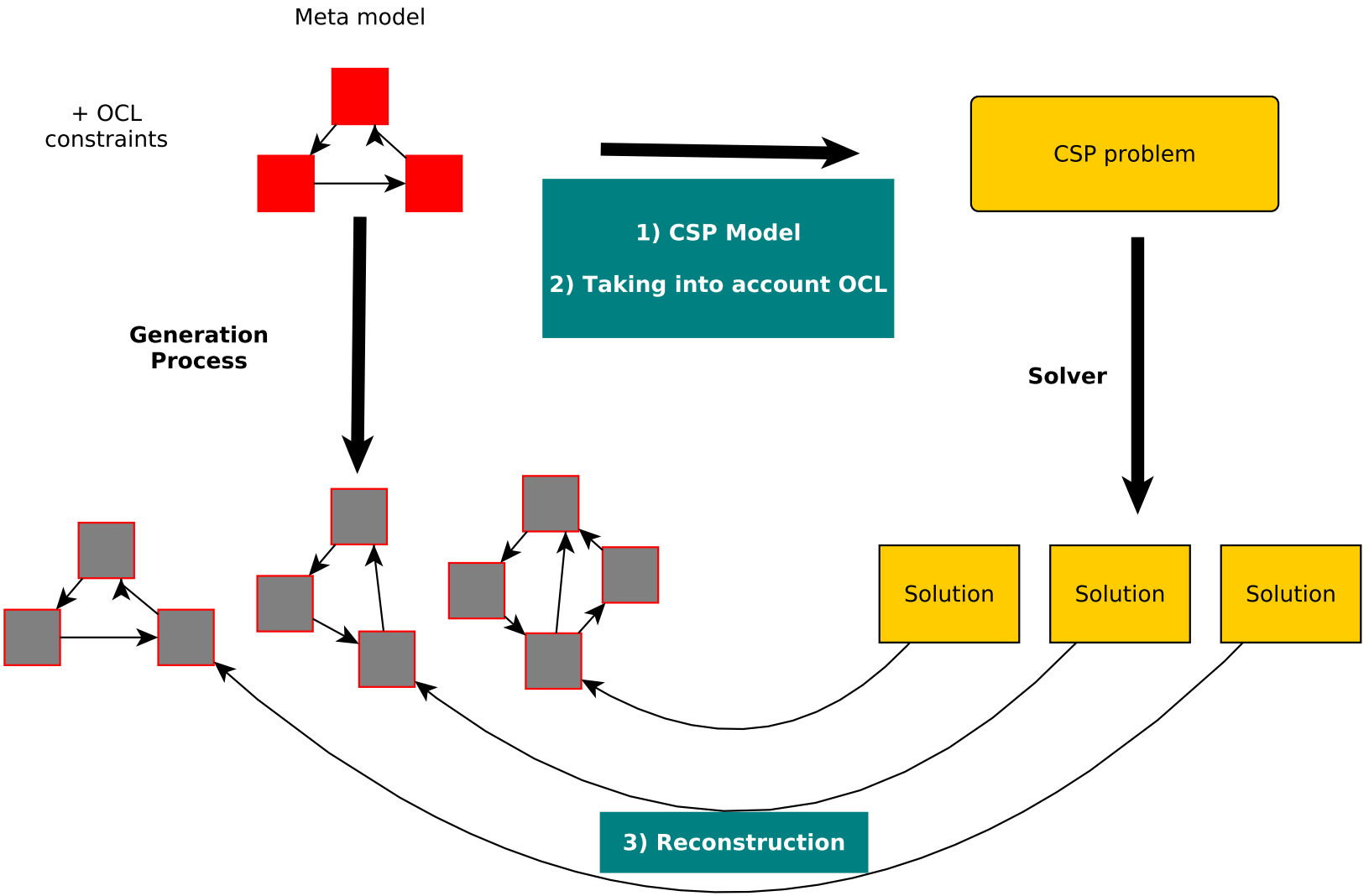
CSP for Model Generation



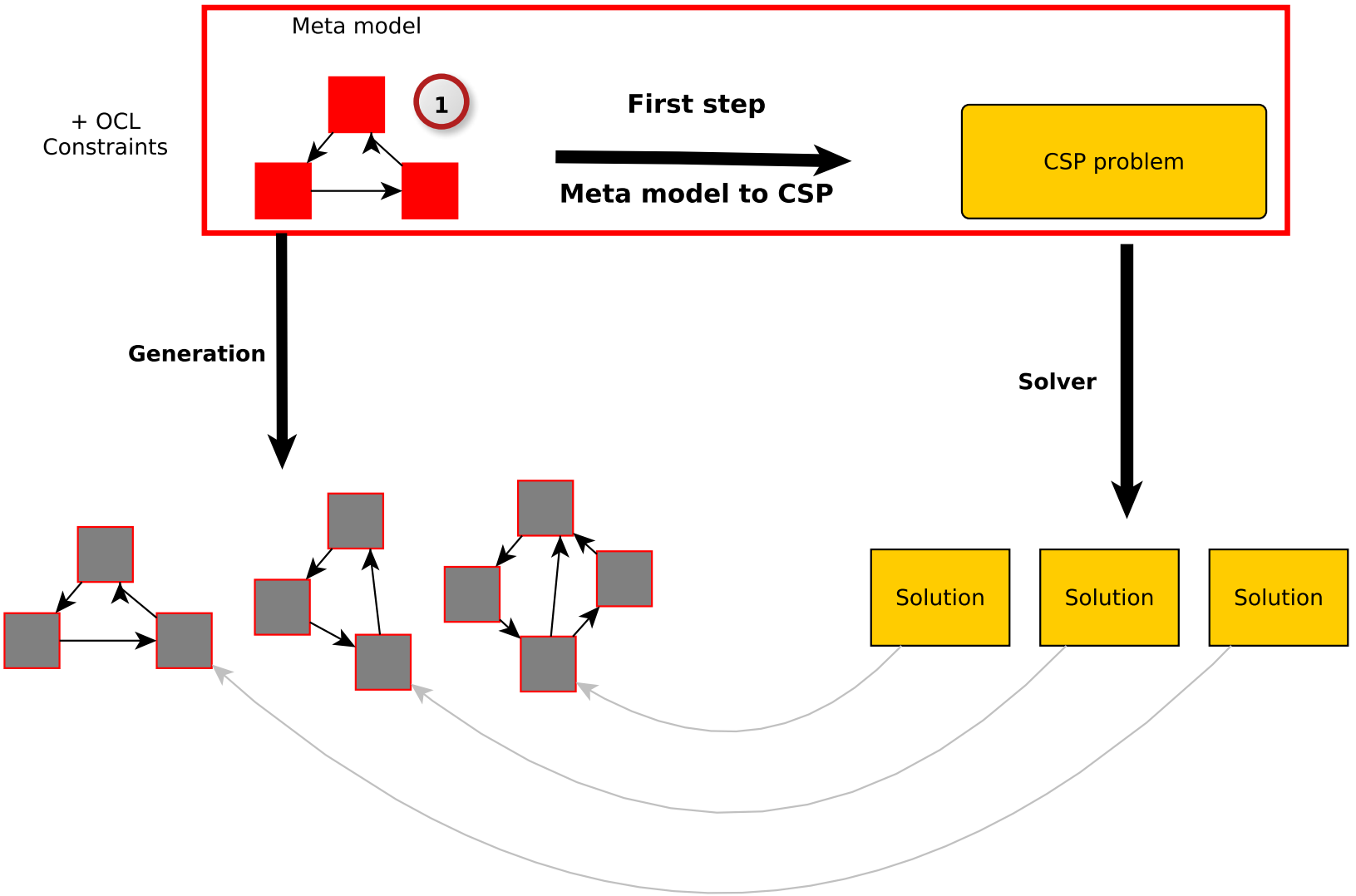
CSP for Model Generation



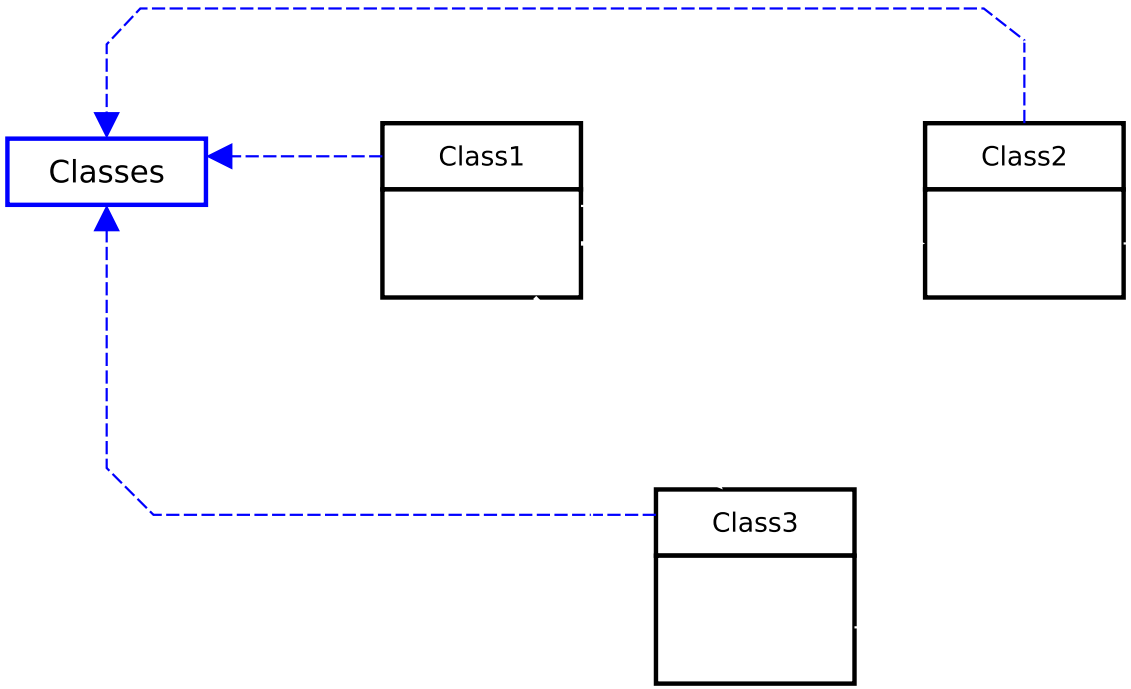
CSP for Model Generation



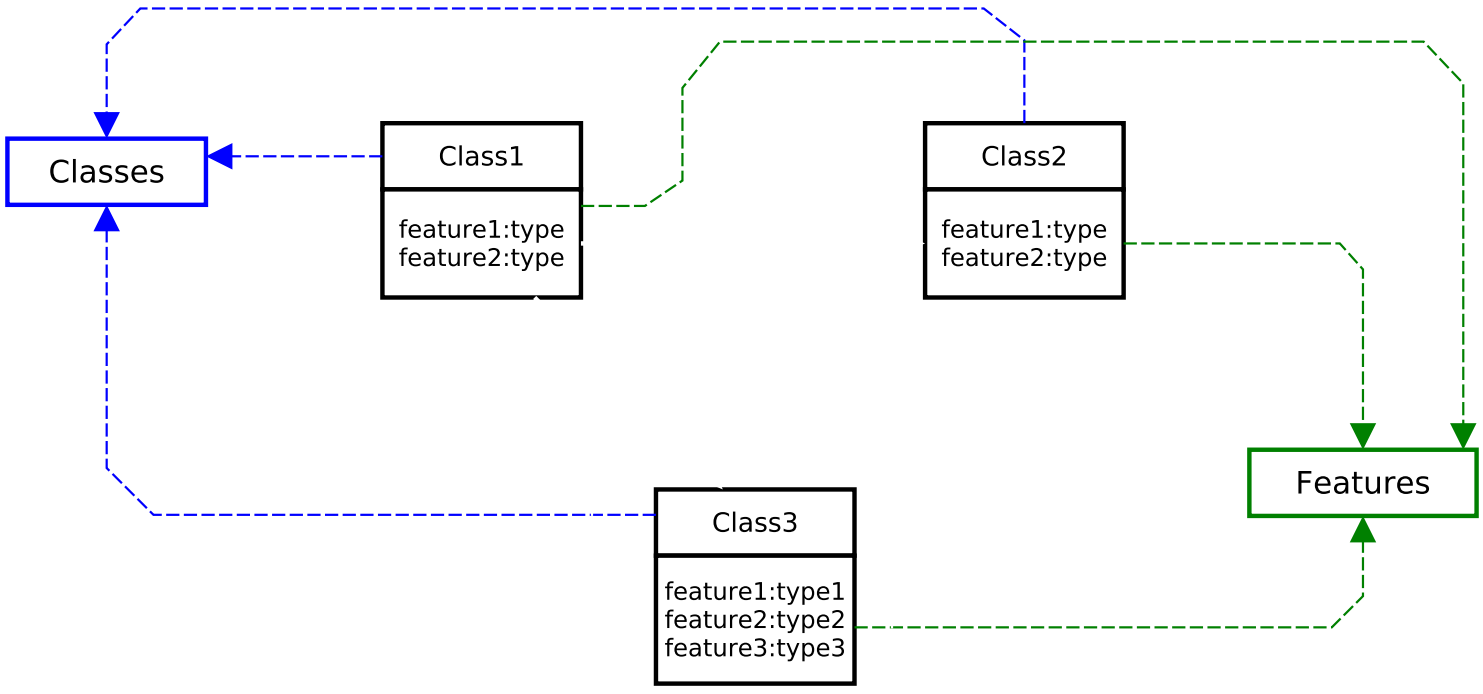
First step: Meta Model to CSP transformation



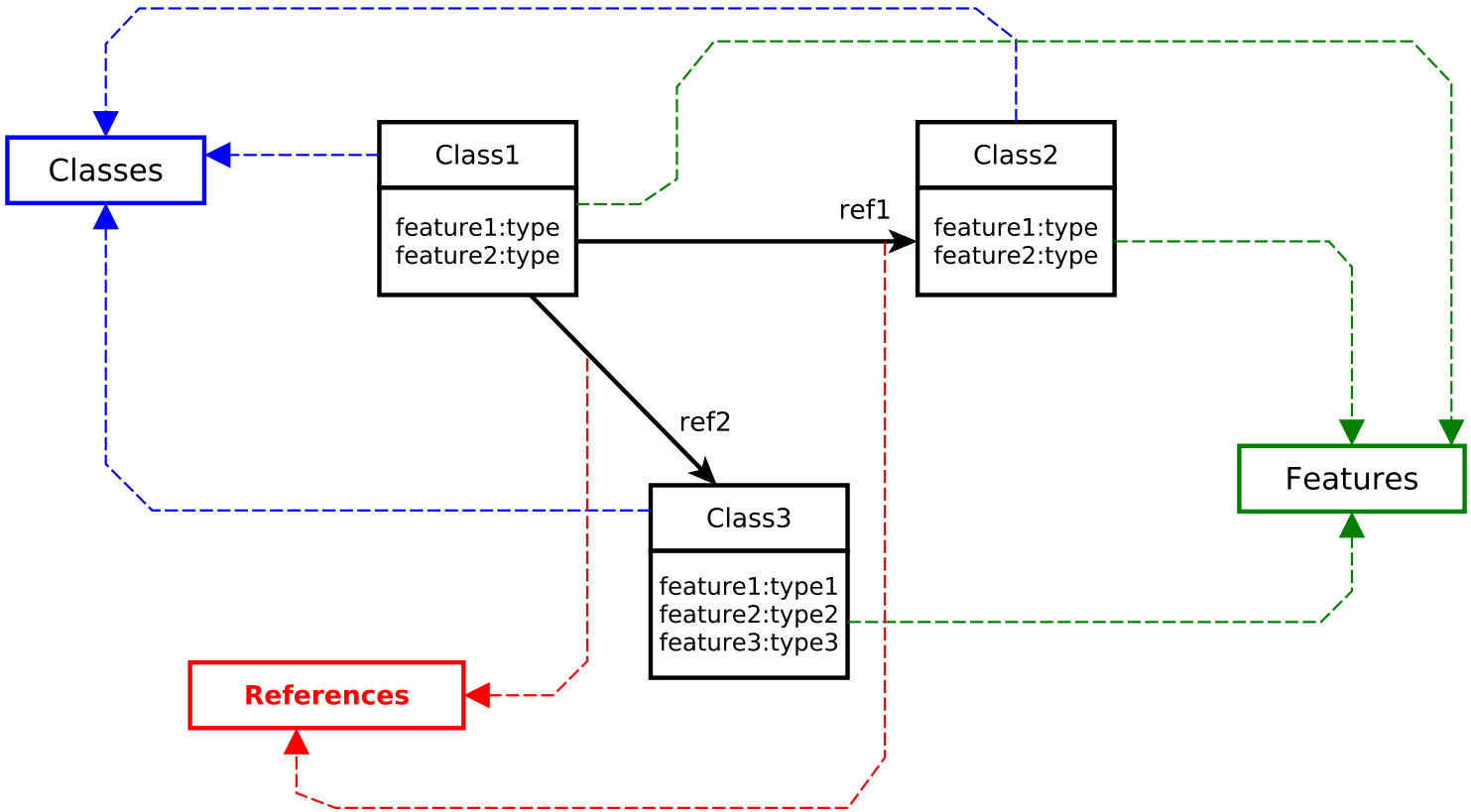
Elements of a meta model



Elements of a meta model



Elements of a meta model



Classes modelling

Class Instances

- One variable per instance.
- Domains including 0 for non-instantiated elements.
- Contiguous domains to distinguish between different classes.
- Root class has only one instance.

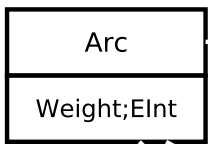
Features

- One variable per each instance feature.

Constraints

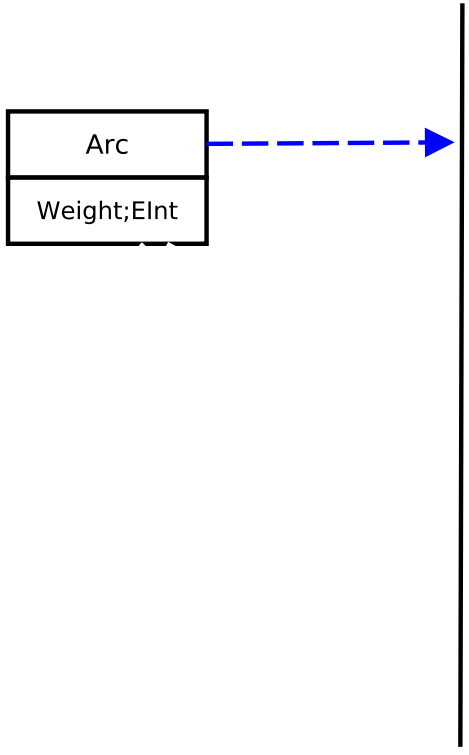
- Gcc on all classes instances to make all variables different except when it is 0.

Classes modelling: Example



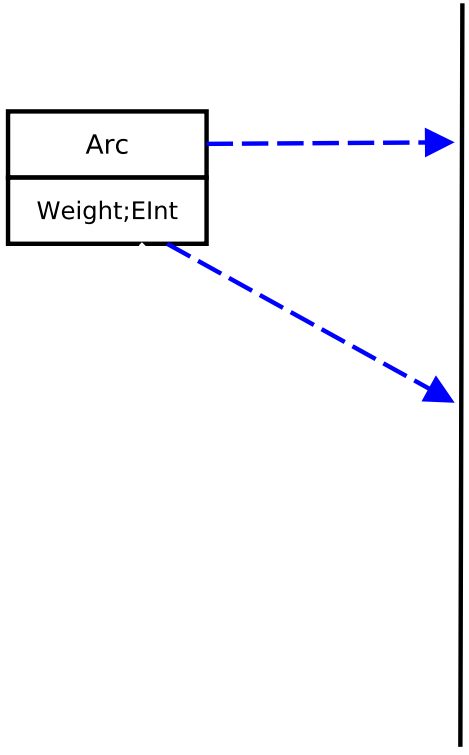
- n instance variables.
  $(D(Arc) = [0, 1, \ldots, n])$
- n feature variables (Weight).
  $(D(Weight) = [-100, 100])$
- $Gcc($ [variables],
  [values],
  $[0, \ldots, 0]$,
  $[m, 1, \ldots, 1])$

Classes modelling: Example



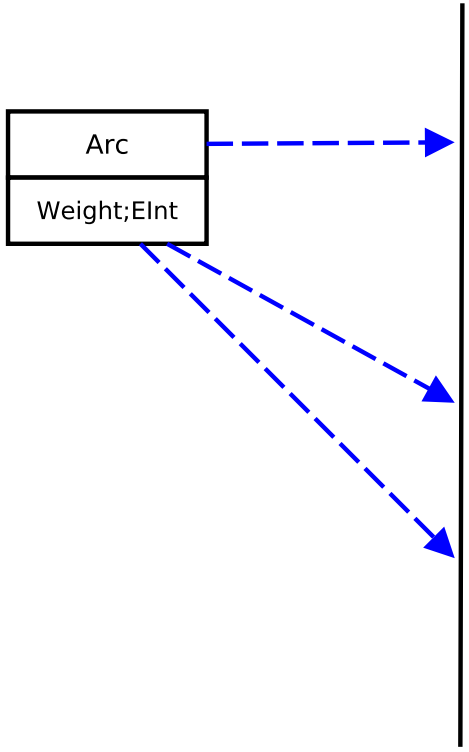
- n instance variables. $(D(Arc) = [0, 1, \ldots, n])$
- n feature variables (Weight). $(D(Weight) = [-100, 100])$
- $Gcc($ [variables], [values], $[0, \ldots, 0]$, $[m, 1, \ldots, 1])$

Classes modelling: Example



- n instance variables.
  $(D(Arc) = [0, 1, \ldots, n])$
- n feature variables (Weight).
  $(D(Weight) = [-100, 100])$
- $Gcc($ [variables],
  [values],
  $[0, \ldots, 0]$,
  $[m, 1, \ldots, 1])$

Classes modelling: Example



- n instance variables.
  $(D(Arc) = [0, 1, \ldots, n])$
- n feature variables (Weight).
  $(D(Weight) = [-100, 100])$
- $Gcc($ [variables],
  [values],
  $[0, \ldots, 0]$,
  $[m, 1, \ldots, 1])$

Reference Modelling

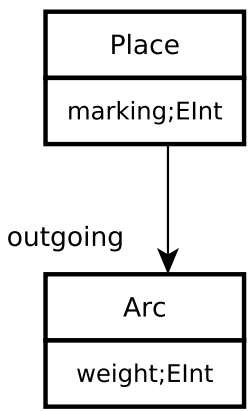
Variables

- Let $r(source \rightarrow target)$ a reference between two classes *source* and *target*.
- Pointer variables from *source* to *target*.
- $D(r) = D(target)$.

Constraints

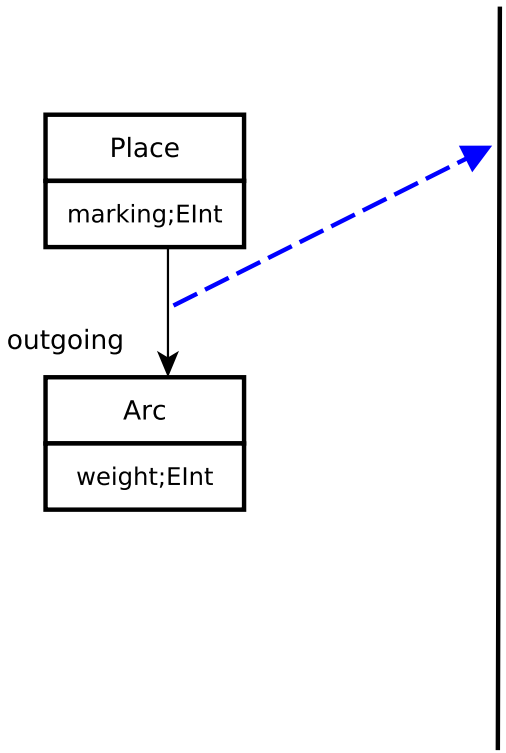
- One Gcc constraint on root class of the meta model.
- One constraint: $source = 0 \rightarrow \text{Reference} = 0$.

Reference Modelling: Example



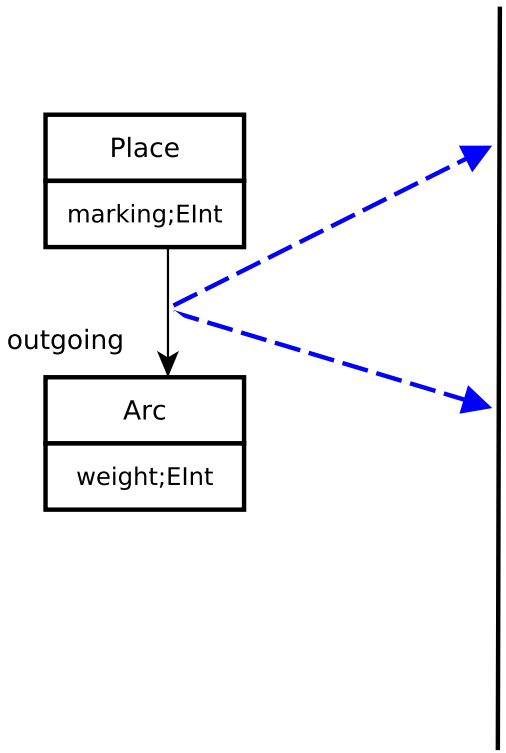
- "pointer" variables from Place to Arc.
  $(D(outgoing) = D(Arc))$
- One Gcc constraint on meta model root class references.
- One constraint:
  $Place = 0 \rightarrow outgoing = 0$.

Reference Modelling: Example



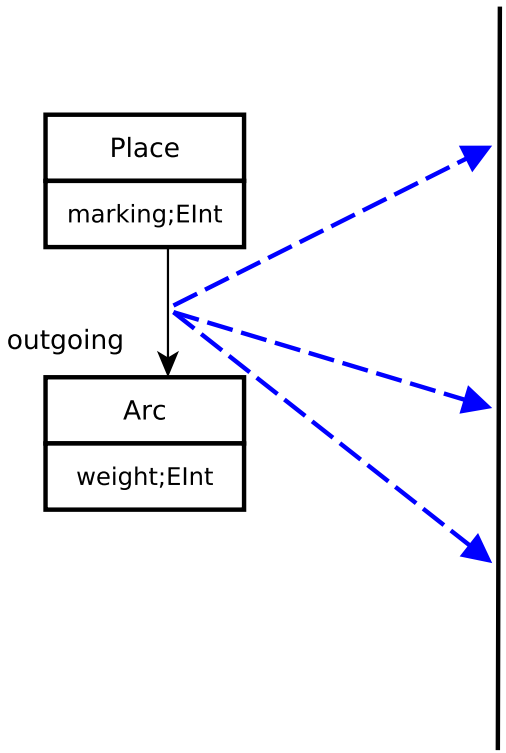
- "pointer" variables from Place to Arc.
  $\big(D(outgoing) = D(Arc)\big)$
- One Gcc constraint on meta model root class references.
- One constraint:
  $Place = 0 \rightarrow outgoing = 0$.

Reference Modelling: Example



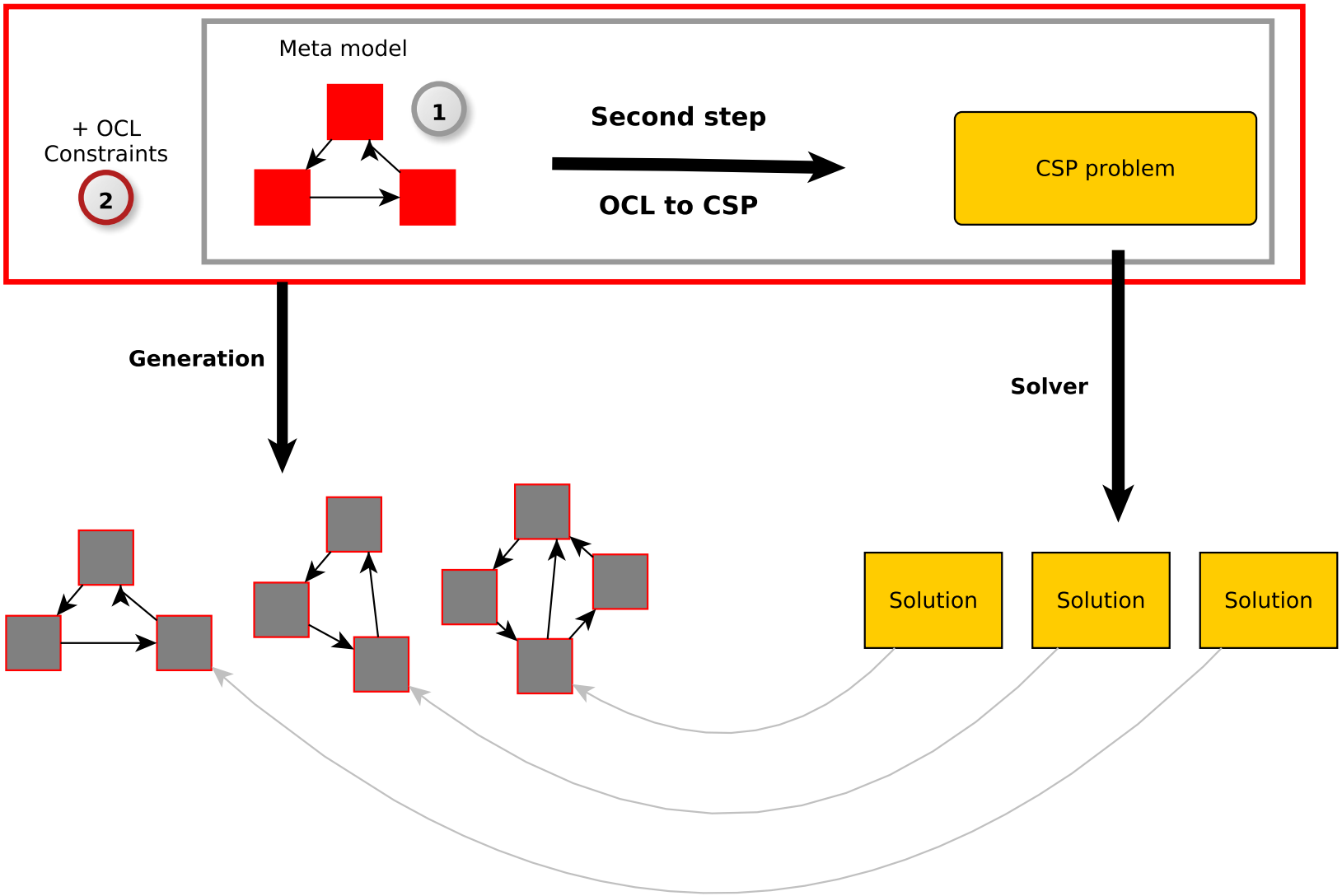
- "pointer" variables from Place to Arc.
  $\big(D(outgoing) = D(Arc)\big)$
- One Gcc constraint on meta model root class references.
- One constraint:
  $Place = 0 \rightarrow outgoing = 0.$

Reference Modelling: Example



- "pointer" variables from Place to Arc.
  $\big(D(outgoing) = D(Arc)\big)$
- One Gcc constraint on meta model root class references.
- One constraint:
  $Place = 0 \rightarrow outgoing = 0$.

Second step: OCL to CSP transformation



OCL constraints modelling

OCL constraints

- OCL (Object Constraints Language) is a formal language based on predicates logic [?].
- Express constraints on meta models elements.
- Widespread OCL constructions are treated in our solution.
- Treatment of references as pointers makes modelling of OCL constraints easier.

OCL constraints modelling

OCL constraints

- OCL (Object Constraints Language) is a formal language based on predicates logic [?].
- Express constraints on meta models elements.
- Widespread OCL constructions are treated in our solution.
- Treatment of references as pointers makes modelling of OCL constraints easier.

OCL constraints modelling

OCL constraints

- OCL (Object Constraints Language) is a formal language based on predicates logic [?].
- Express constraints on meta models elements.
- Widespread OCL constructions are treated in our solution.
- Treatment of references as pointers makes modelling of OCL constraints easier.

OCL constraints modelling

OCL constraints

- OCL (Object Constraints Language) is a formal language based on predicates logic [?].
- Express constraints on meta models elements.
- Widespread OCL constructions are treated in our solution.
- Treatment of references as pointers makes modelling of OCL constraints easier.

OCL constraint about class feature

CSP model

- Limit the domain of the variable representing the feature.
- Apply the boolean expression of the OCL constraint on previous domain values to obtain the new domain.

Consequence ...

- No new CSP variables created.
- No new CSP constraints created.
- Only domains treatment.

OCL constraint about class feature

CSP model

- Limit the domain of the variable representing the feature.
- Apply the boolean expression of the OCL constraint on previous domain values to obtain the new domain.

Consequence ...

- No new CSP variables created.
- No new CSP constraints created.
- Only domains treatment.

OCL constraint modelling: Example

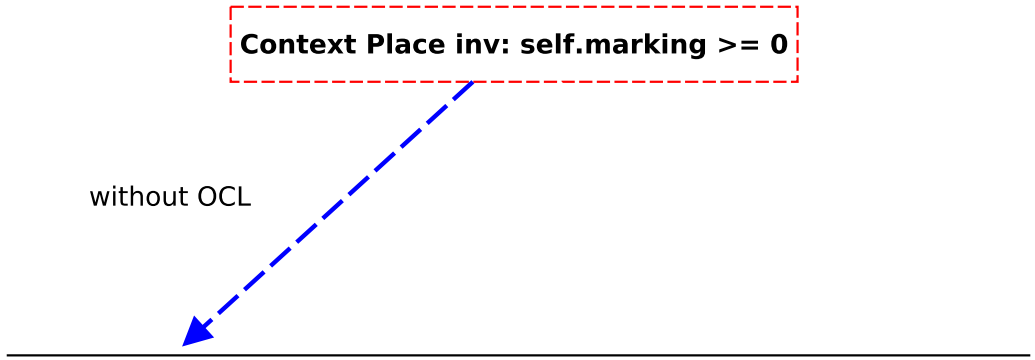
Context Place inv: self.marking >= 0

$D(marking) = [-100, 100]$

$D(marking) = [0, 100]$

Neither variables nor constraints created.

OCL constraint modelling: Example

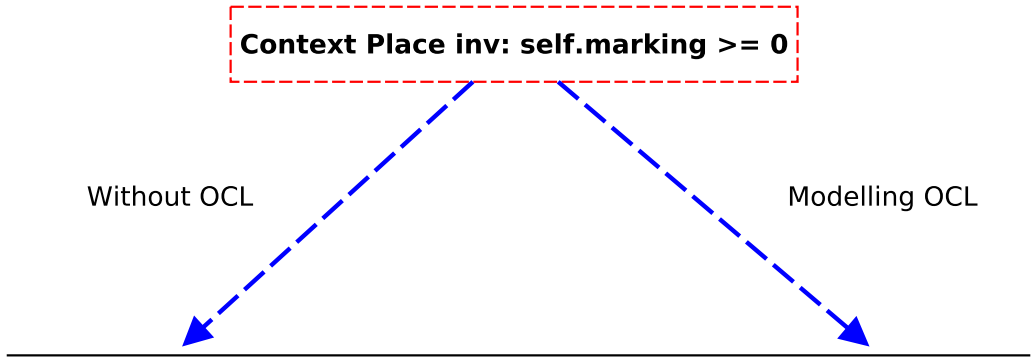


$D(marking) = [-100, 100]$

$D(marking) = [0, 100]$

Neither variables nor constraints created.

OCL constraint modelling: Example



$D(marking) = [-100, 100]$

$D(marking) = [0, 100]$

Neither variables nor constraints created.

Improvements provided by our solution I

Solution of Cabot et al.

Use of list variables to distinguish instance variables of different classes.

Our solution

A set of variables with contiguous domains is used to differentiate between classes.

Reminder

Cabot and al. are the authors of an existing CSP approach.

Improvements provided by our solution II

Solution of Cabot et al.

Absence of global constraints.

Our solution

The use of some global constraints as Gcc reduces the number of constraints.

Improvements provided by our solution III

Solution of Cabot et al.

A reference is considered as a couple of two variables.

Our solution

It is is considered as a pointer from source class to target class. Thus, we reduce the number of variables and make OCL treatment easier.

Improvements provided by our solution IV

Solution of Cabot et al.

Systematic and brutal OCL to CSP transformation.

Our solution

Transformation adapted for each kind of OCL constraints. For instance, we just limit domains for the easiest OCL constraints.

Summary

Experiments purposes

Used Meta models

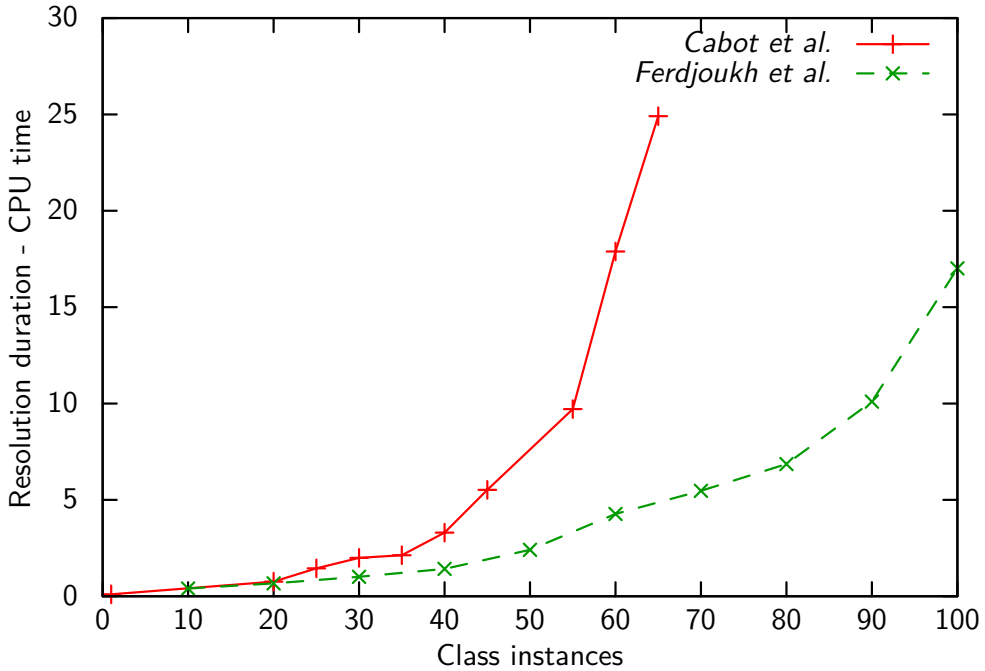
- Petri nets meta model: 4 classes [5].
- Entities and relationships: 5 classes [6].
- B language meta model: 34 classes [?].
- Sad meta model: 40 classes [?].

Experiments purposes

- Compare our solution results with those of *Cabot et al.*
- Verify the scaling of the solution. Show that it is possible to generate models containing an important number of elements.

Experiments I

1. Resolution **8 times quicker** than existing solution.
   - CSP variables number reduced.
   - Efficient treatment of class references.

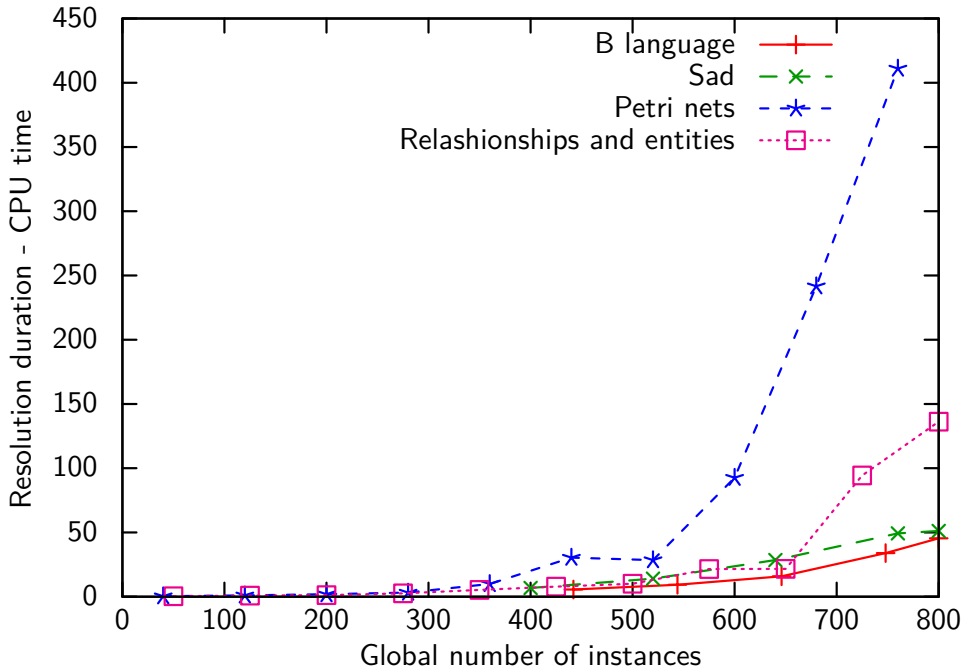


Experiments made on Entities and Relationships meta models.

Experiments II

2. Good scaling (both models and meta models).
   - Not a lot of constraints.
   - Use of Global Constraints like Gcc.

Meta model	#Classes
Petri nets	4
Enti. and Relat.	5
B language	34
Sad	40



Summary

Conclusion

Model Generation

- An important issue in MDE.
- CSP approach to solve this problem.
- Taking into account OCL constraints.

Results

- The experiments show that our solution is more efficient than existing solution.
- It scales well as size of generated models or meta models increases.

Future works

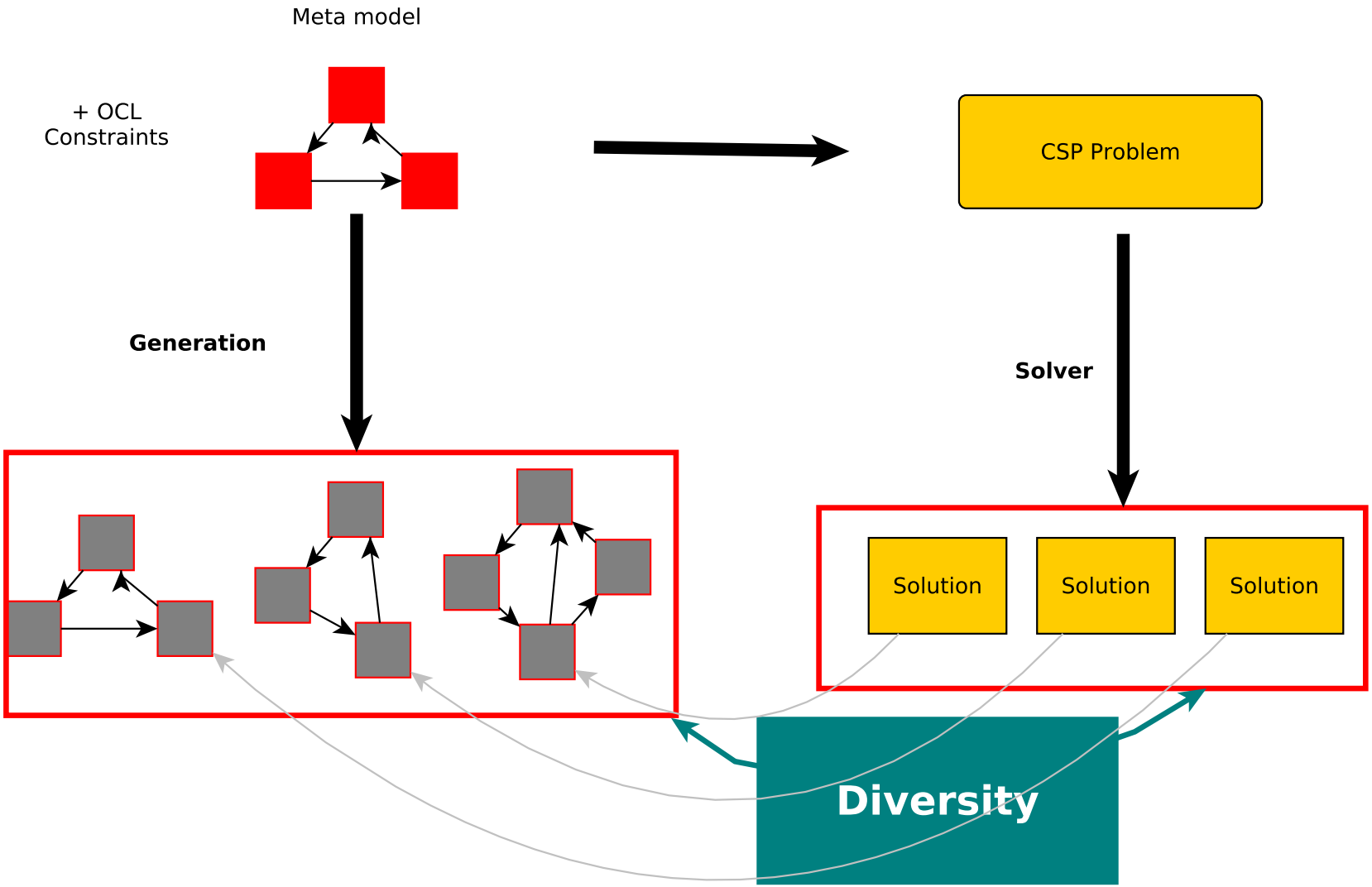
Improve scaling

- Do reification of implication constraints.
- Break symmetries on references.

OCL constraints

Model all OCL constructions.

Future works: Solutions Diversity



Tanemirt-nwen

References I

Several model generation paradigms

Model Generation approaches

- *Sen and al.* : Alloy [**?**, **?**].
- *Ehrig and al.* : Graph grammar [**?**].
- *Mougenot and al.* : Random graph generation [**?**].
- *Cabot and al.* : CSP [**?**, **?**].

So why do we choose a CSP approach ?

- Lack of flexibility of other approaches.
- Other approaches do not take into account OCL constraints.
- The existing CSP approach is not efficient.

```
index("Paper", 1).

% Position of attribute wordCount
% within the list of attributes
attIndex("Paper", "wordCount", 2).

nodeConstant(_, _, Result):-
   Result = 10000.

nodeVariable(_, Vars, Result ):-          % x = var of the innermost iterator
   nth1(1, Vars, Result).                 % Result = Vars[1] = value of x

nodeAttrib(Instances, Vars, Result):-
   nodeVariable(Instances, Vars, Object), % An object of class Paper
   attIndex("Paper", "wordCount", N),     % N = Index of field wordCount
   arg(N, Object, Result).                % Result = Object[N] = wordCount value

nodeAllInstances(Instances, Vars, Result) :-
   index("Paper", N),                     % N = Position of class Paper
   nth1(N, Instances, Result).            % Result = Instances[N] = Inst of Paper

nodeLessThan(Instances, Vars, Result) :-
   nodeAttrib(Instances, Vars, Value1),   % 1st subexpression
   nodeConstant(Instances, Vars, Value2), % 2nd subexpression
   #<(Value1, Value2, Result).            % Result = (Value1 < Value2)?

nodeForAll(Instances, Vars, Result) :-
   nodeAllInstances(Instances, Vars, L),  % L = Result of allInstances
   ( foreach(Elem, L), foreach(Eval, Out), param(Instances,Vars) do
       % Eval = Result of evaluating nodeLessThan on an element of L
       nodeLessThan(Instances, [Elem|Vars], Eval) ),
   % Out = List of truth values.  Out[i]= Result of nodeLessThan(L[i])
   length(L, N),                          % N = length(L)
   #=(N, sum(Out), Result).               % Result = (N = ΣOut[i])?
```

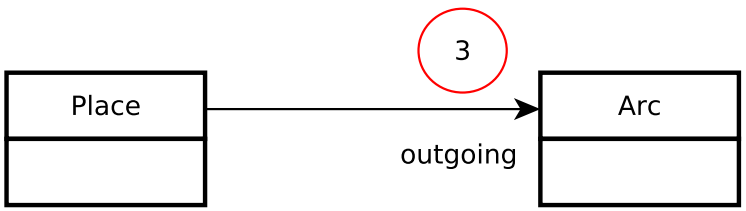
Reference Navigation OCL constraint

Context Place inv: self.outgoing.traitement()

- Modelling easier because of pointers.
- Apply treatment to all variables respecting the equality.
- In the paper, generalized to n references.

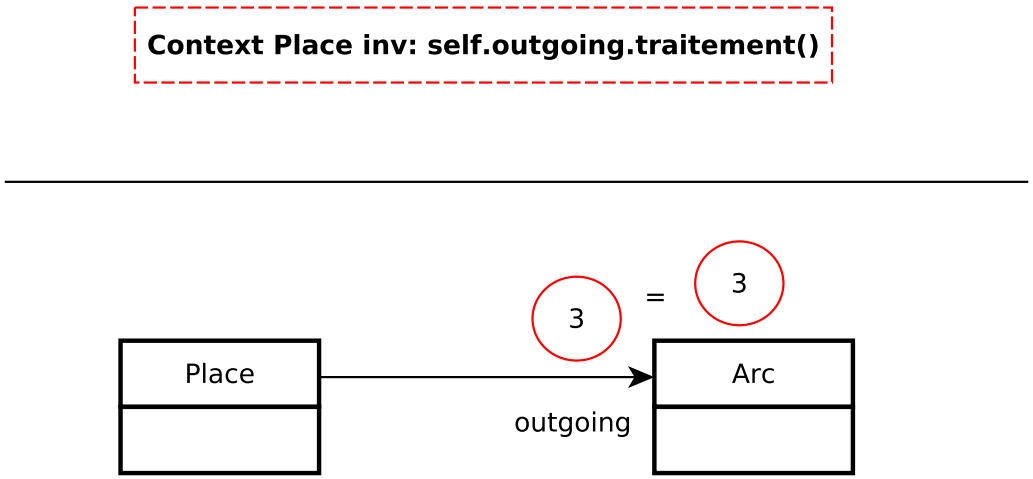
Reference Navigation OCL constraint

Context Place inv: self.outgoing.traitement()



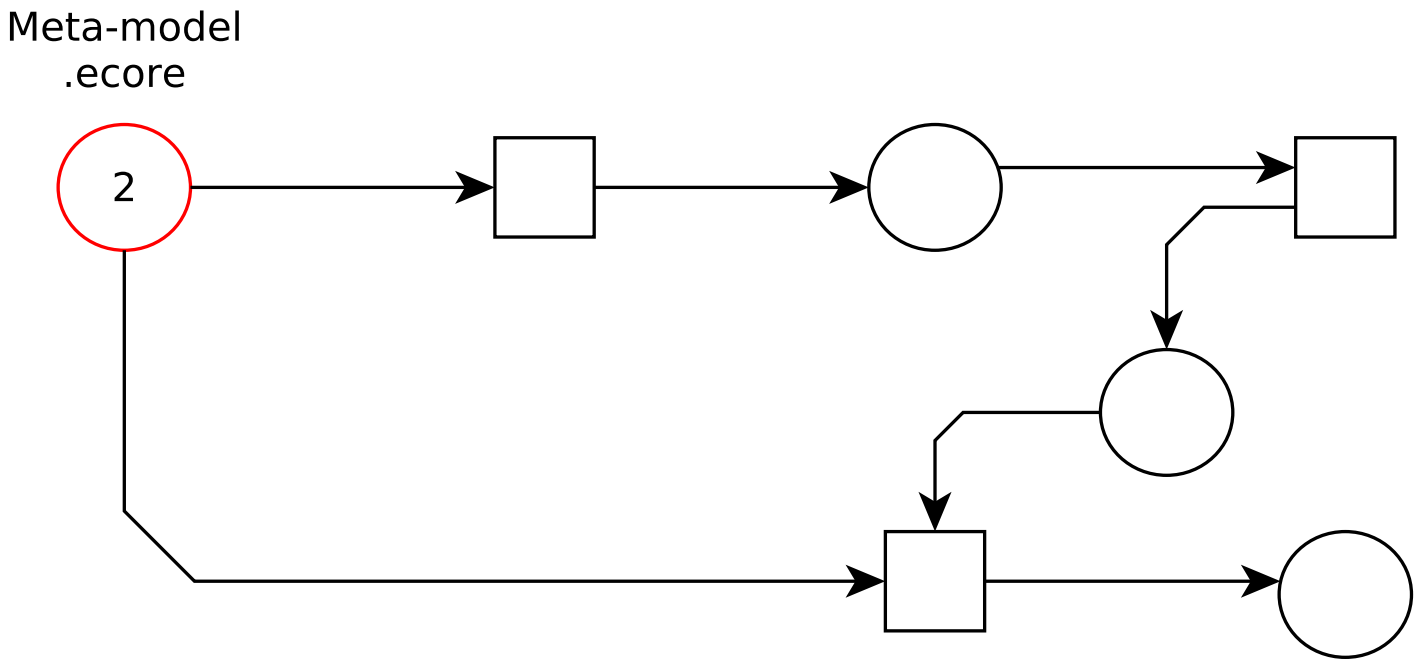
- Modelling easier because of pointers.
- Apply treatment to all variables respecting the equality.
- In the paper, generalized to *n* references.

Reference Navigation OCL constraint



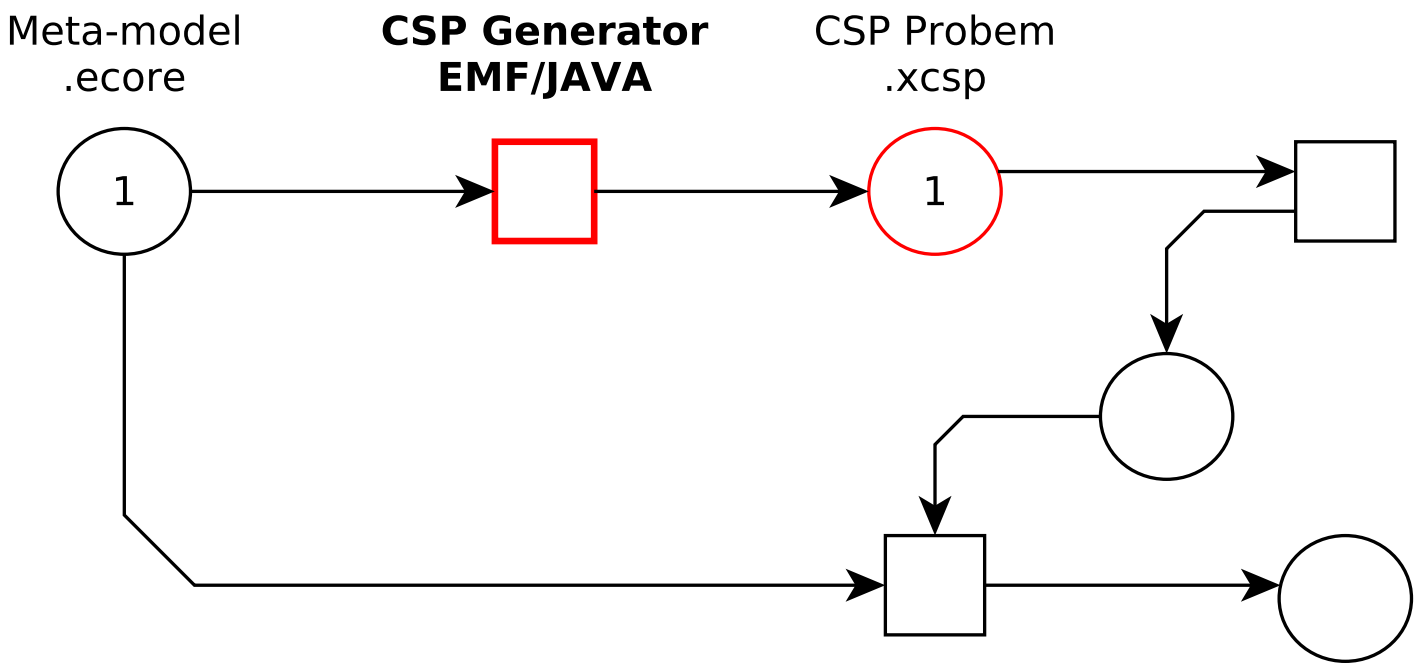
- Modelling easier because of pointers.
- Apply treatment to all variables respecting the equality.
- In the paper, generalized to *n* references.

Implementation



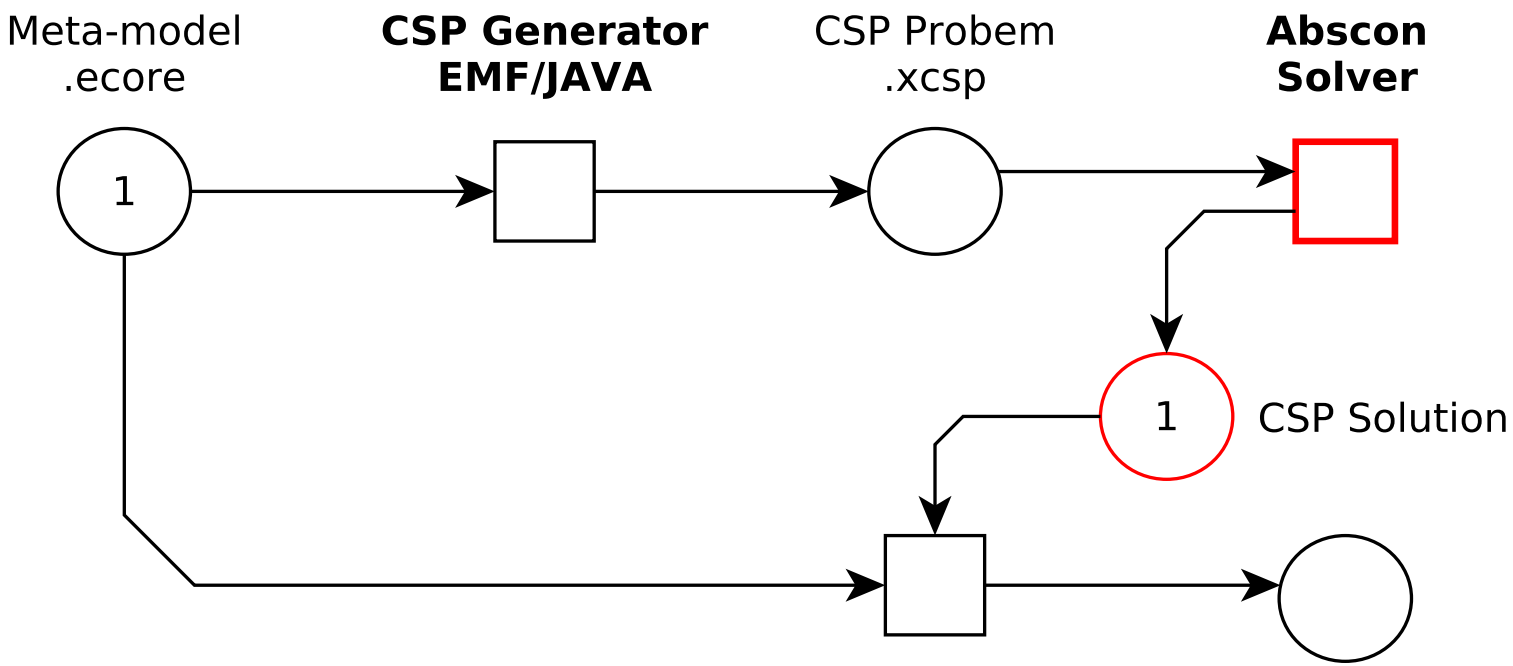
[?].

Implementation



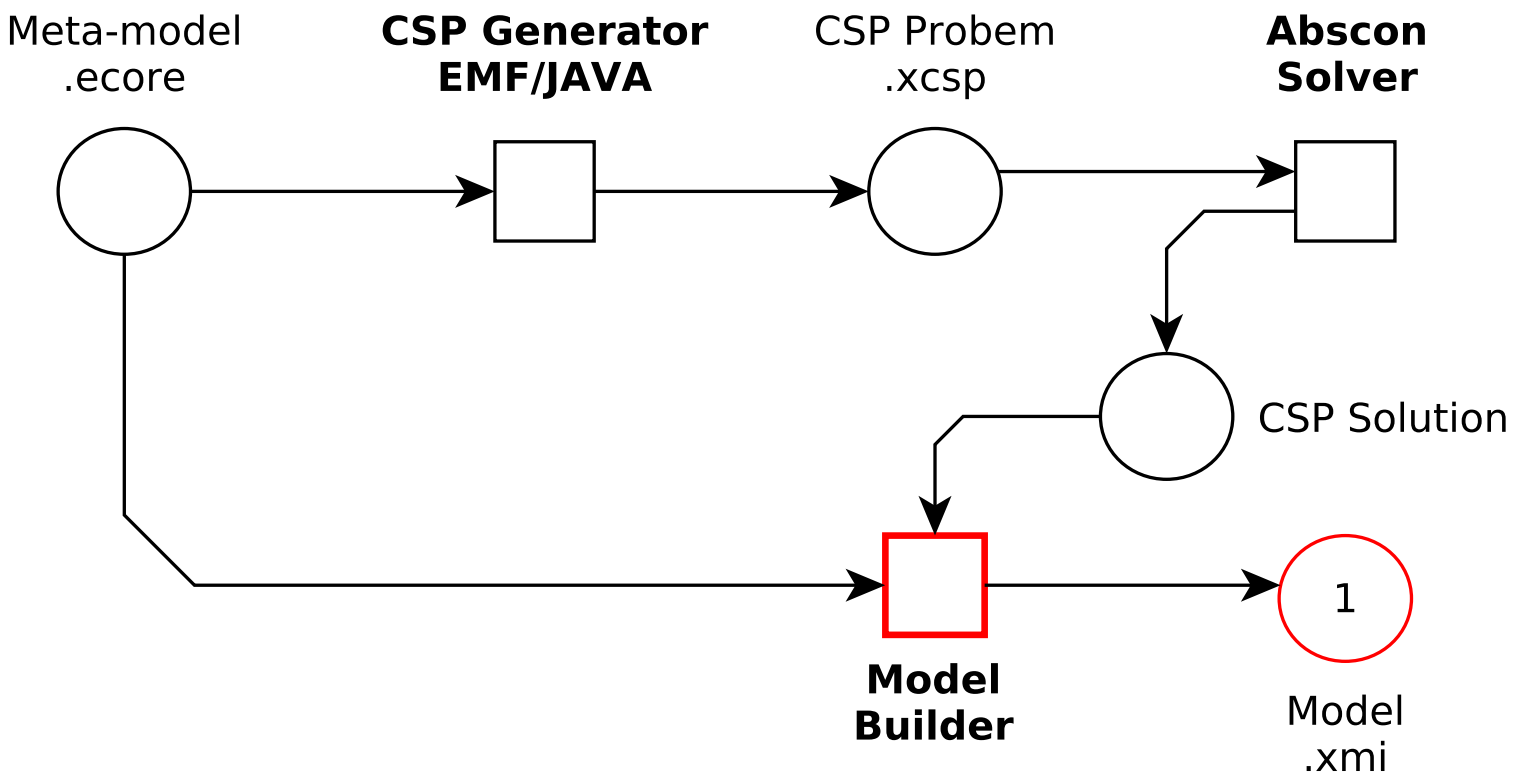
[?].

Implementation



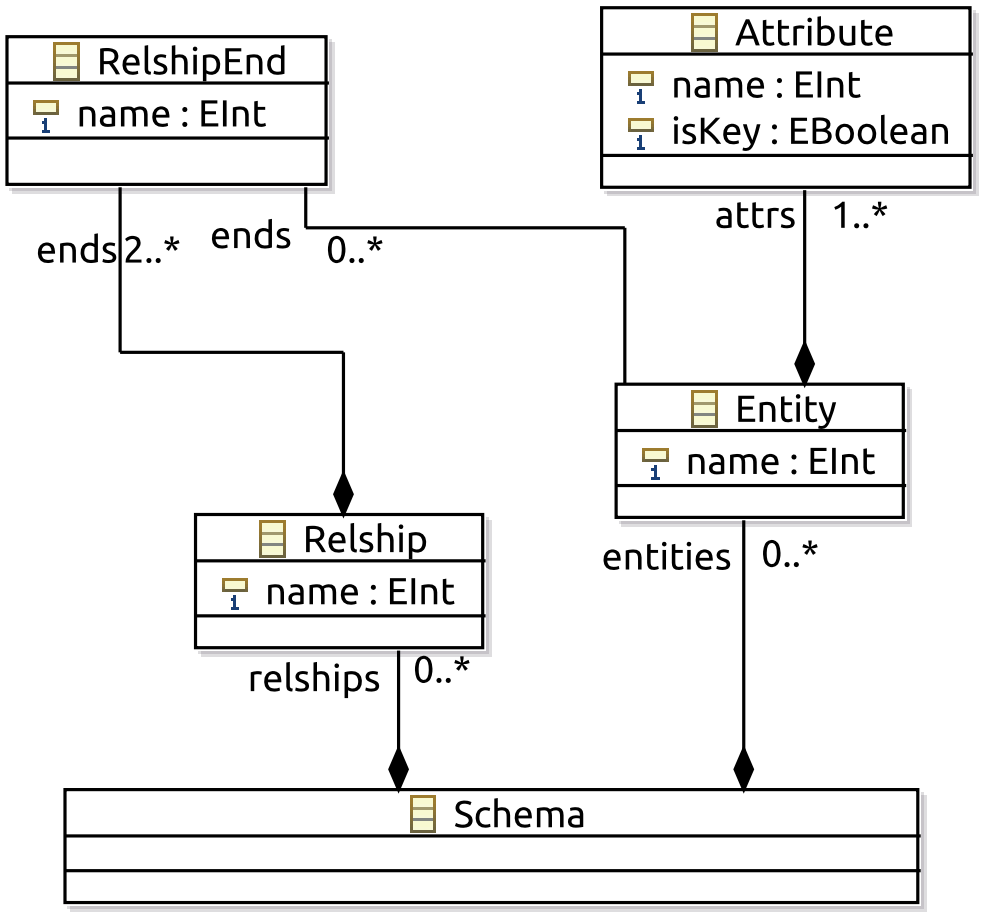
[?].

Implementation



[?].

Entities and relationships meta model



*