Using ATL to define advanced and flexible constraint model transformations

Raphaël Chenouard[1] Laurent Granvilliers[1] Ricardo Soto[1,2]

[1]CNRS, LINA, Université de Nantes, France

[2]Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso, Chile

MTATL 2009, Nantes, France.

8-queens problem

Placing eight chess queens on an 8x8 chessboard such that none of them is able to capture any other using the standard chess queen's moves.

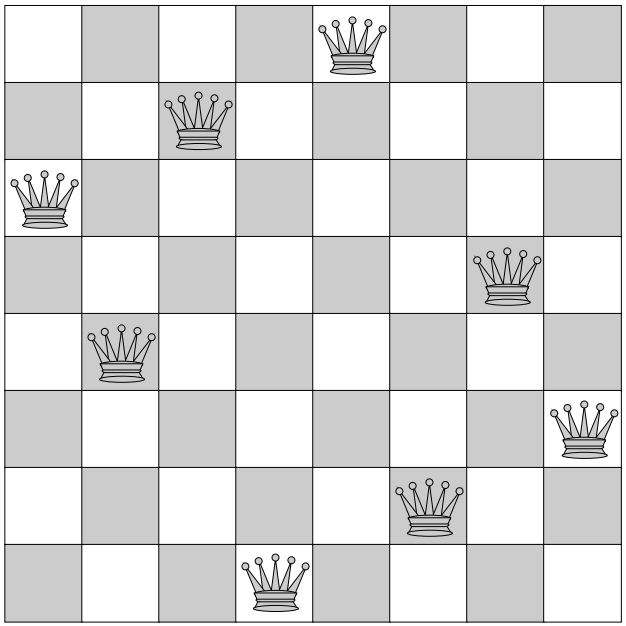
Variables

One for each queen.

Constraints

To avoid two queens sharing the same:

- Row.
- SW-NE diagonal.
- NW-SE diagonal.



8-queens problem

Placing eight chess queens on an 8x8 chessboard such that none of them is able to capture any other using the standard chess queen's moves.

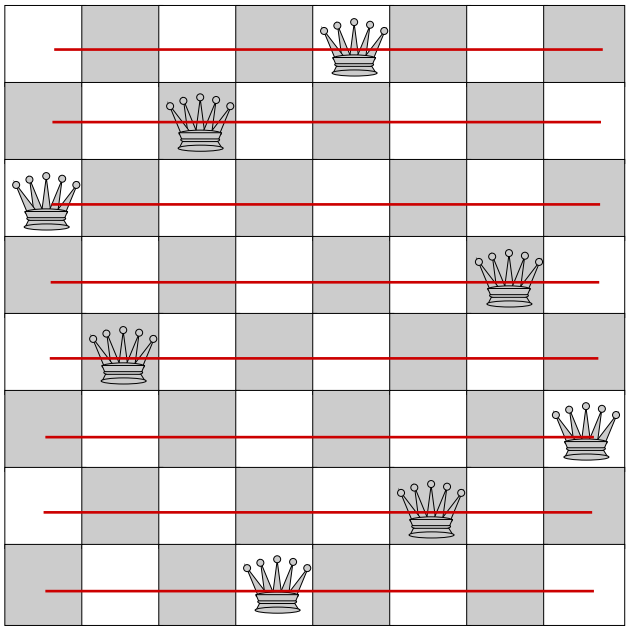
Variables

One for each queen.

Constraints

To avoid two queens sharing the same:

- Row.
- SW-NE diagonal.
- NW-SE diagonal.



8-queens problem

Placing eight chess queens on an 8x8 chessboard such that none of them is able to capture any other using the standard chess queen's moves.

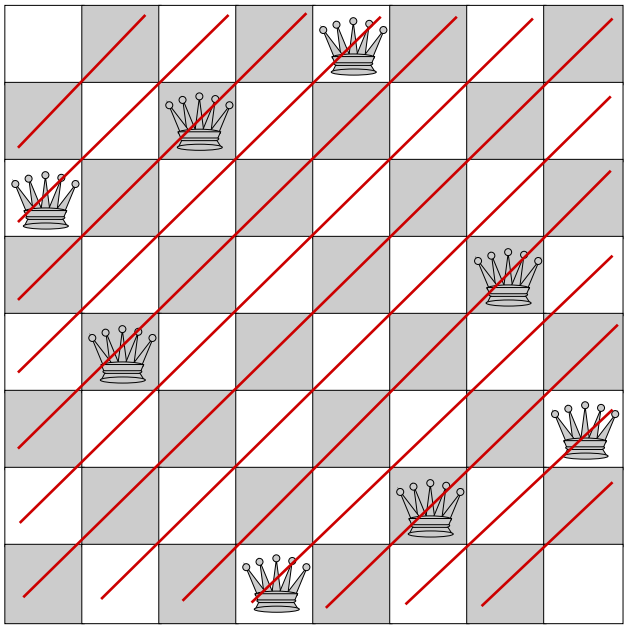
Variables

One for each queen.

Constraints

To avoid two queens sharing the same:

- Row.
- SW-NE diagonal.
- NW-SE diagonal.



8-queens problem

Placing eight chess queens on an 8x8 chessboard such that none of them is able to capture any other using the standard chess queen's moves.

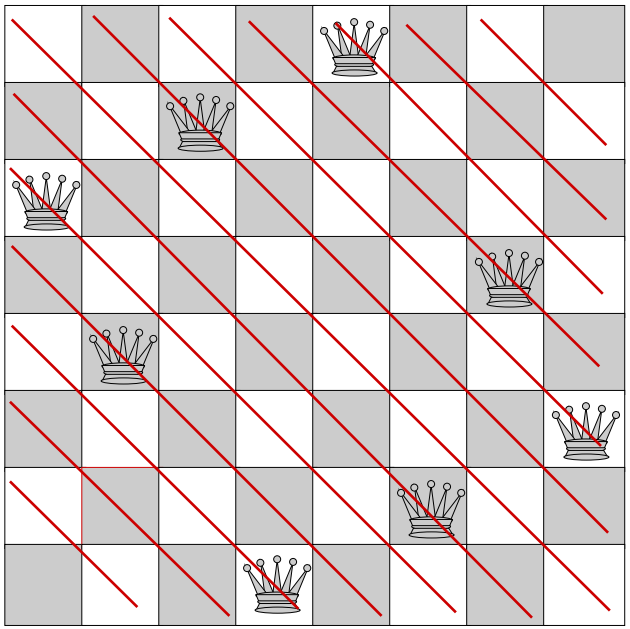
Variables

One for each queen.

Constraints

To avoid two queens sharing the same:

- Row.
- SW-NE diagonal.
- NW-SE diagonal.



Existing CP languages

Several kinds of languages:

- **CLP (Constraint Logic Programming)**
  ECL^{i}PS^e, GNU Prolog, SICStus Prolog...
- **Libraries**
  ILOG Solver (C++), Gecode (C++), Koalog (Java), CHOCO (Java)...
- **Modeling languages**
  MiniZinc, OPL, Alice...

Several kinds of users:

- CP solver experts : directly using solvers and their constructs,
- CP beginner (but expert in the studied domain) : high level modeling with only a few solver constructs.

High-level model and low-level (solver) model

```
//Modèle OPL
int n = 8;
range Domain 1..n;
var Domain board[Domain];
solve {
  forall(ordered i,j in Domain) {
    board[i]     <> board[j];
    board[i] + i <> board[j] + j;
    board[i] - i <> board[j] - j;
  };
};
```

```
package examples;
import static org.gecode.Gecode.*;
import static org.gecode.GecodeEnumConstants.*;
import org.gecode.*;

public class Queens extends Space {
    public VarArray<IntVar> board;

    public Queens(Options opt) {
        super();
        int n = opt.size;
        board = new VarArray<IntVar>(this, n, IntVar.class, 1, n);

        for(int i=0;i<=n-1;i++) {
            for(int j=i+1;j<=n-1;j++) {
                post(this, new Expr().p(board.get(i)),IRT_NQ,
                    new Expr().p(board.get(j)));
                post(this, new Expr().p(board.get(i)).p(i),IRT_NQ,
                    new Expr().p(board.get(j)).p(j));
                post(this, new Expr().p(board.get(i)).m(i),IRT_NQ,
                    new Expr().p(board.get(j)).m(j));
            }
         }
         branch(this, board, BVAR_SIZE_MIN, BVAL_MIN);
    }

    public Queens(Boolean share, Queens queens) {
        super(share, queens);
        board = new VarArray<IntVar>(this, share, queens.board);
    }

    public String toString() {
        int i;
        String st = "";
        for (i=0;i<board.size();i++){
            if(board.get(i).assigned())
                st += board.get(i).val() + " ";
        }
        return st;
    }

    public static void main(String[] args) {
        Options opt = new Options();
        opt.size = 8;
        opt.gui = true;
        opt.parse(args);
        opt.name = "Queens";
        Queens queens = new Queens(opt);
        opt.doSearch(queens);
    }
}
```

High-level model and low-level (solver) model

10 code lines

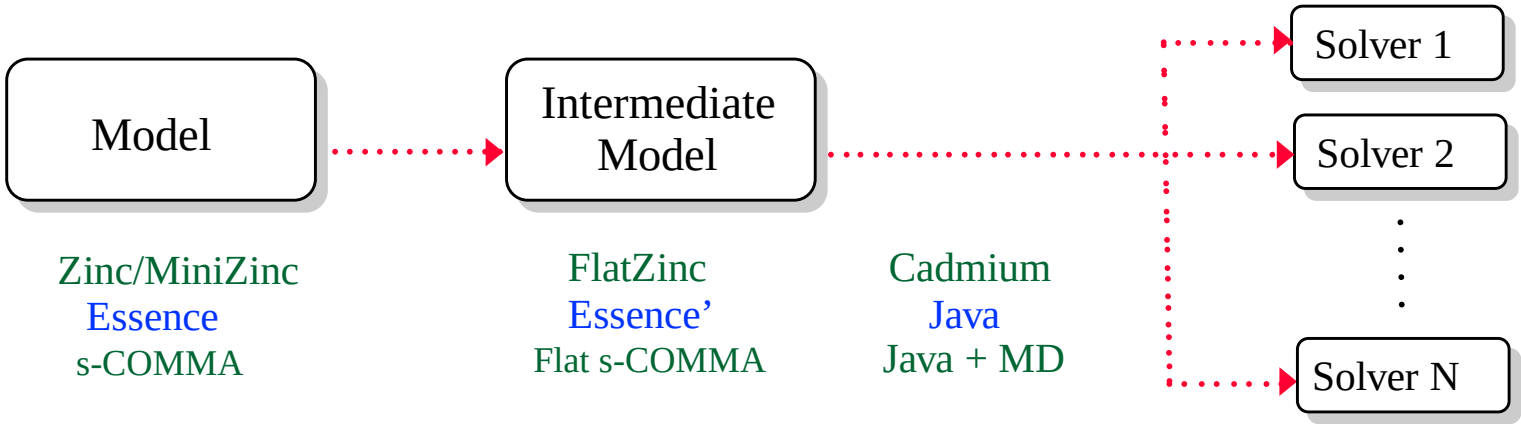
```
//Modèle OPL
int n = 8;
range Domain 1..n;
var Domain board[Domain];
solve {
  forall(ordered i,j in Domain) {
    board[i]     <> board[j];
    board[i] + i <> board[j] + j;
    board[i] - i <> board[j] - j;
  };
};
```

51 code lines

```
//Modèle Gecode/J
...
public Queens(Options opt) {
  super();
  int n = opt.size;
  board = new VarArray<IntVar>
         (this, n, IntVar.class, 1, n);

  for(int i=0;i<=n-1;i++) {
    for(int j=i+1;j<=n-1;j++) {
      post(this, new Expr().p(board.get(i)),
               IRT_NQ,
               new Expr().p(board.get(j)));
      post(this, new Expr().p(board.get(i)).p(i),
               IRT_NQ,
               new Expr().p(board.get(j)).p(j));
      post(this, new Expr().p(board.get(i)).m(i),
               IRT_NQ,
               new Expr().p(board.get(j)).m(j));
    }
  }
  branch(this, board, BVAR_SIZE_MIN, BVAL_MIN);
}
...
```

Existing modeling platforms

A **one-to-many** architecture with solver independent languages: Zinc/MiniZinc, Essence, s-COMMA.

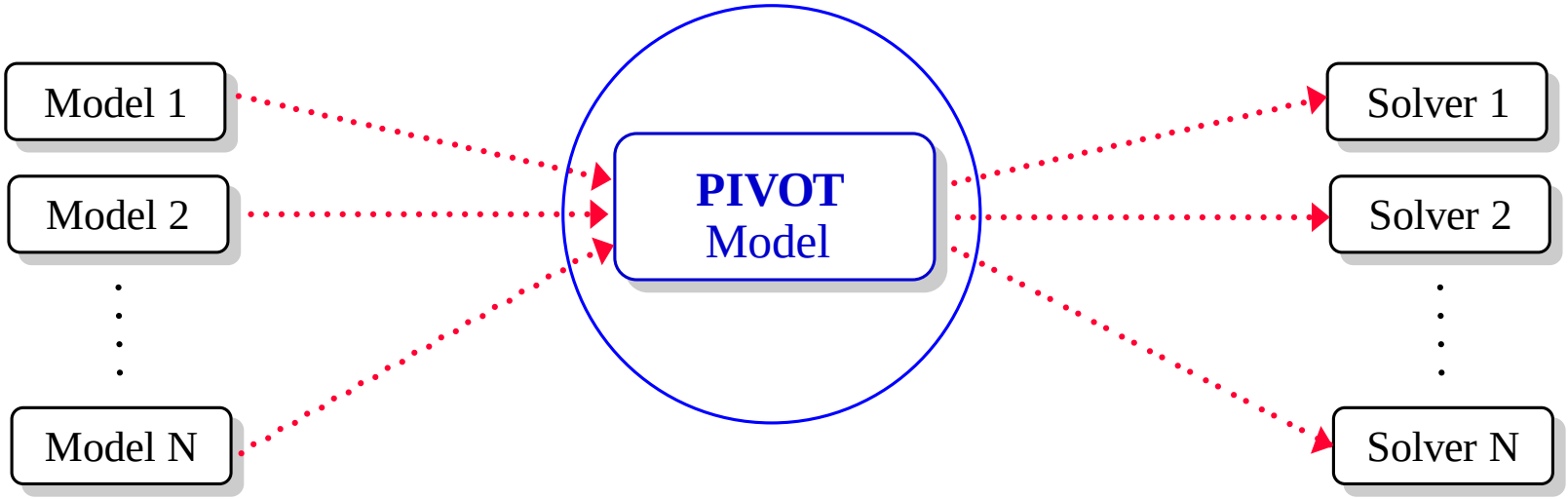


Assets

- The user may choose the best solving technology.
- The platform is open to add new solvers.

A new architecture: Many-to-Many

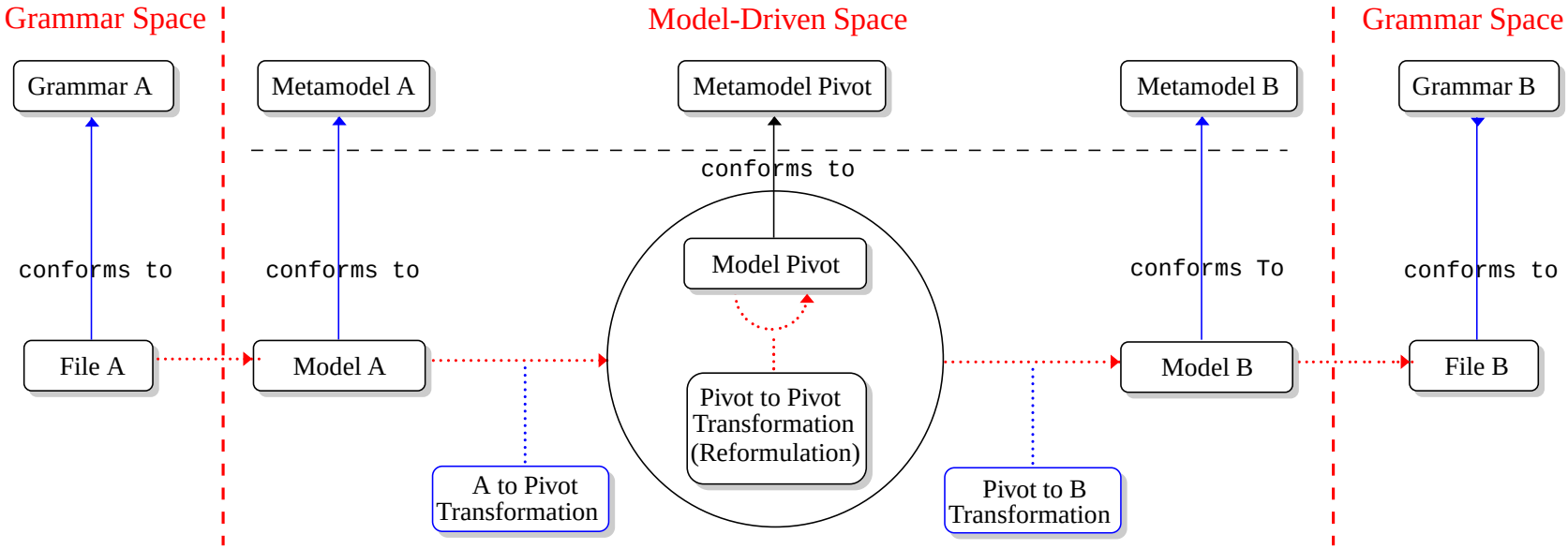
A new architecture independent from solver and modeling languages:



Assets

- The user may choose the best solving and modeling technology.
- The platform is open to add new solver and modeling languages.
- Possibility of selecting the refactoring rules.

A Model-Driven platform



Assets

- Model transformations make use of metamodels semantics.
- Independence from concrete syntaxes.

A motivating example

The social golfers problem

A set of $n = g \times s$ golfers wishes to play golf each week. There are g groups of s golfers. The problem is to find a playing schedule for w weeks such that no two golfers play together more than once.

A solution considering 3 groups of 3 golfers for 4 weeks:

	Group 1	Group 2	Group 3
Week 1	[a, b, c]	[d, e, f]	[g, h, i]
Week 2	[a, d, g]	[b, e, h]	[c, f, i]
Week 3	[a, f, h]	[b, d, i]	[c, e, g]
Week 4	[a, e, i]	[b, f, g]	[c, d, h]

A motivating example

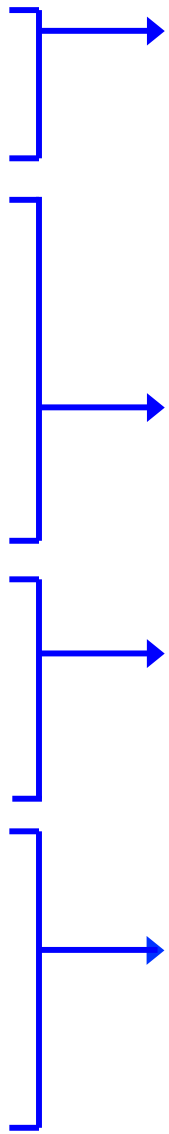
```
enum Name := {a,b,c,d,e,f,g,h,i};
int s := 3; //size of groups
int w := 4; //number of weeks
int g := 3; //groups per week

main class SocialGolfers {
   Week weekSched[w];
   constraint differentGroups {
     forall(w1 in 1..w, w2 in w1+1..w)
       forall(g1 in 1..g, g2 in 1..g)
         card(weekSched[w1].groupSched[g1]
         .players intersect weekSched[w2].
         groupSched[g2]
   }
}
class Group {
  Name set players;
  constraint groupSize {
    card(players) = s;
  }
}
class Week {
  Group groupSched[g];
  constraint playOncePerWeek {
    forall(g1 in 1..g, g2 in g1+1..g)
      card(groupSched[g1].players
      intersect
      groupSched[g2].players) = 0;
  }
}
```

s-COMMA ▶ **PIVOT** ▶ ECL^{i}PS^e

```
socialGolfers(L):-
S $= 3,W $= 4,G $= 3,

intsets(WEEKSCHED_GROUPSCHED_PLAYERS,12,1,9),
L = WEEKSCHED_GROUPSCHED_PLAYERS,

(for(W1,1,W),param(L,W,G) do
 (for(W2,W1+1,W),param(L,G,W1) do
  (for(G1,1,G),param(L,G,W1,W2) do
   (for(G2,1,G),param(L,G,W1,W2,G1) do
    V1 is G*(W1-1)+G1,nth(V2,V1,L),
    V3 is G*(W2-1)+G2,nth(V4,V3,L),
                      V5),V5 $=< 1
                                  ram(L,S,W,G) do
 (for(I2,1,G),param(L,S,W,G,I1) do
  V6 is G*(I1-1)+I2,nth(V7,V6,L),
  #(V7, V8), V8 $= S
 )
),
(for(I1,1,W),param(L,G) do
 (for(G1,1,G),param(L,G,I1) do
  (for(G2,G1+1,G),param(L,G,I1,G1) do
   V9 is G*(I1-1)+G1,nth(V10,V9,L),
   V11 is G*(I1-1)+G2,nth(V12,V11,L),
   #(V10 /\ V12, 0)
 ))
),
label_sets(L).
```

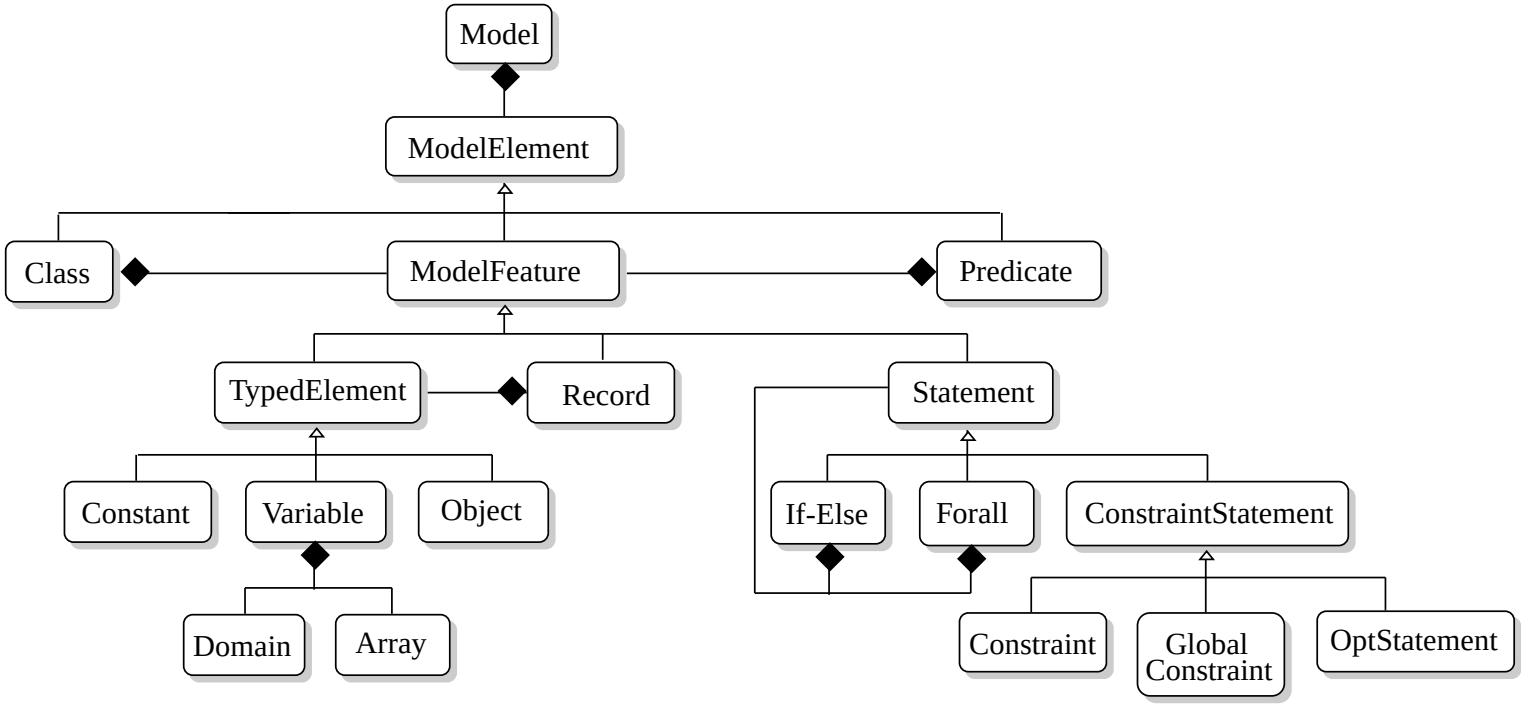
A motivating example

```
enum Name := {a,b,c,d,e,f,g,h,i};
int s := 3; //size of groups
int w := 4; //number of weeks
int g := 3; //groups per week

main class SocialGolfers {
   Week weekSched[w];
   constraint differentGroups {
     forall(w1 in 1..w, w2 in w1+1..w)
       forall(g1 in 1..g, g2 in 1..g)
         card(weekSched[w1].groupSched[g1]
         .players intersect weekSched[w2].
         groupSched[g2].players) <= 1;
   }
}
class Group {
  Name set players;
  constraint groupSize {
    card(players) = s;
  }
}
class Week {
  Group groupSched[g];
  constraint playOncePerWeek {
    forall(g1 in 1..g, g2 in g1+1..g)
      card(groupSched[g1].players
      intersect
      groupSched[g2].players) = 0;
  }
}
```



```
socialGolfers(L):-
S $= 3,W $= 4,G $= 3,

intsets(WEEKSCHED_GROUPSCHED_PLAYERS,12,1,9),
L = WEEKSCHED_GROUPSCHED_PLAYERS,

(for(W1,1,W),param(L,W,G) do
 (for(W2,W1+1,W),param(L,G,W1) do
  (for(G1,1,G),param(L,G,W1,W2) do
   (for(G2,1,G),param(L,G,W1,W2,G1) do
    V1 is G*(W1-1)+G1,nth(V2,V1,L),
    V3 is G*(W2-1)+G2,nth(V4,V3,L),
    #(V2 /\ V4, V5),V5 $=< 1
 )))
),
(for(I1,1,W),param(L,S,W,G) do
 (for(I2,1,G),param(L,S,W,G,I1) do
  V6 is G*(I1-1)+I2,nth(V7,V6,L),
  #(V7, V8), V8 $= S
 )
),
(for(I1,1,W),param(L,G) do
 (for(G1,1,G),param(L,G,I1) do
  (for(G2,G1+1,G),param(L,G,I1,G1) do
   V9 is G*(I1-1)+G1,nth(V10,V9,L),
   V11 is G*(I1-1)+G2,nth(V12,V11,L),
   #(V10 /\ V12, 0)
 ))
),
label_sets(L).
```

Pivot Metamodel



- Capture most of CP concepts.
- Reformulation steps are only performed on Pivot models.

Choosing reformulation steps

Available reformulation steps:

- Objects flattening
- Enumeration removal
- Loop unrolling
- If removal
- Expression simplification
- ...

```
rule CSPModel {
  from
    s : PivotCSP!CSPModel
  to
    t : PivotCSP!CSPModel (
      name <- s.name,
      elements <- s.elements->
        union(s.elements->select(r|
        r.oclIsTypeOf(PivotCSP!CSPRecord)
      )->collect(r|
        r.getAllElements
      )->flatten())
    )
}
```

Chaining reformulation steps

- Each chain can be described and customized in a script.
- Reformulation steps may be chosen following source model concepts and target model concepts.

Discussion

Supported languages:

- s-COMMA, MiniZinc(in progress), ECL^{i}PS^e, Gecode/J, Realpaver.

Translation times from s-COMMA to ECL^{i}PS^e:

Problems	Time (in s)
SocialGolfers	0.771
Engine	1.166
SendMoreMoney	0.651
Stable Marriage	0.951
10-Queens	0.564

Pivot metamodel flexibility

- Adding new modeling concepts in the pivot is easy.
- New reformulation steps can be added.

$\Rightarrow$ adding new modeling or solver languages becomes easier.

Conclusion future work

A new architecture for rewriting constraint models:

- 3 main steps: source $\longrightarrow$ pivot $- \cdots \longrightarrow$ pivot $\longrightarrow$ target.
- Pivot models are independent from modeling and solving languages.
- Reformulation steps are generic and applied to pivot models.
- Flexible chaining of reformulation steps.

Future work:

- Integrating new languages in the platform (Essence, MiniZinc).
- Automating the choice of reformulation steps.