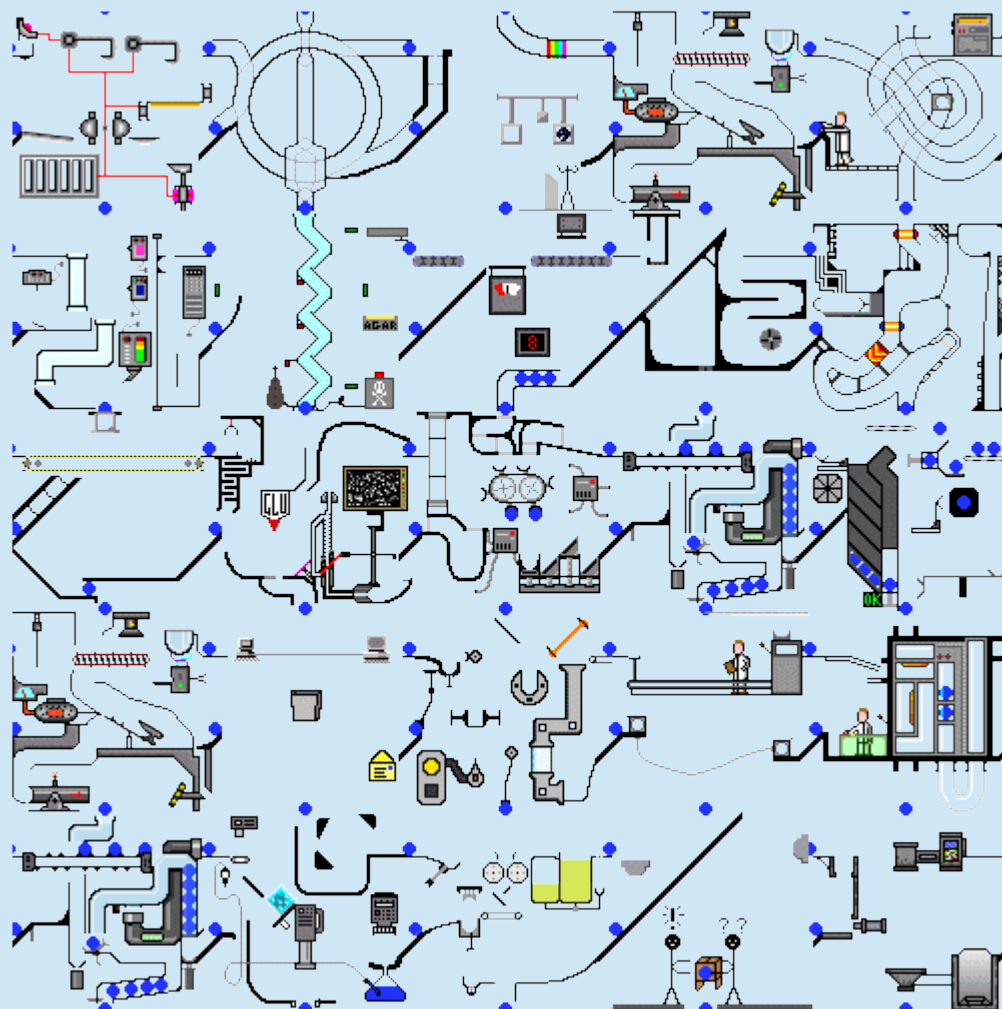
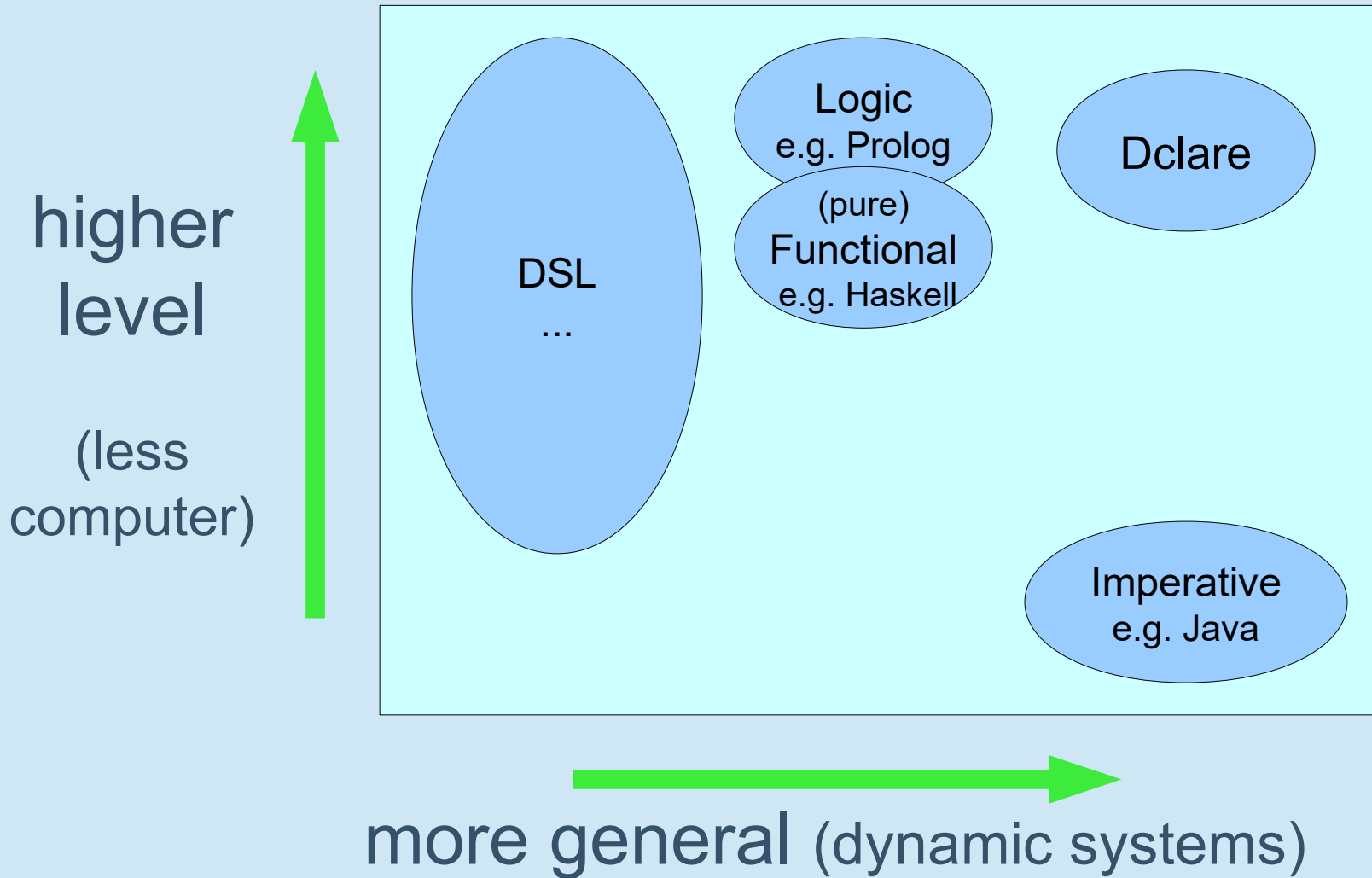


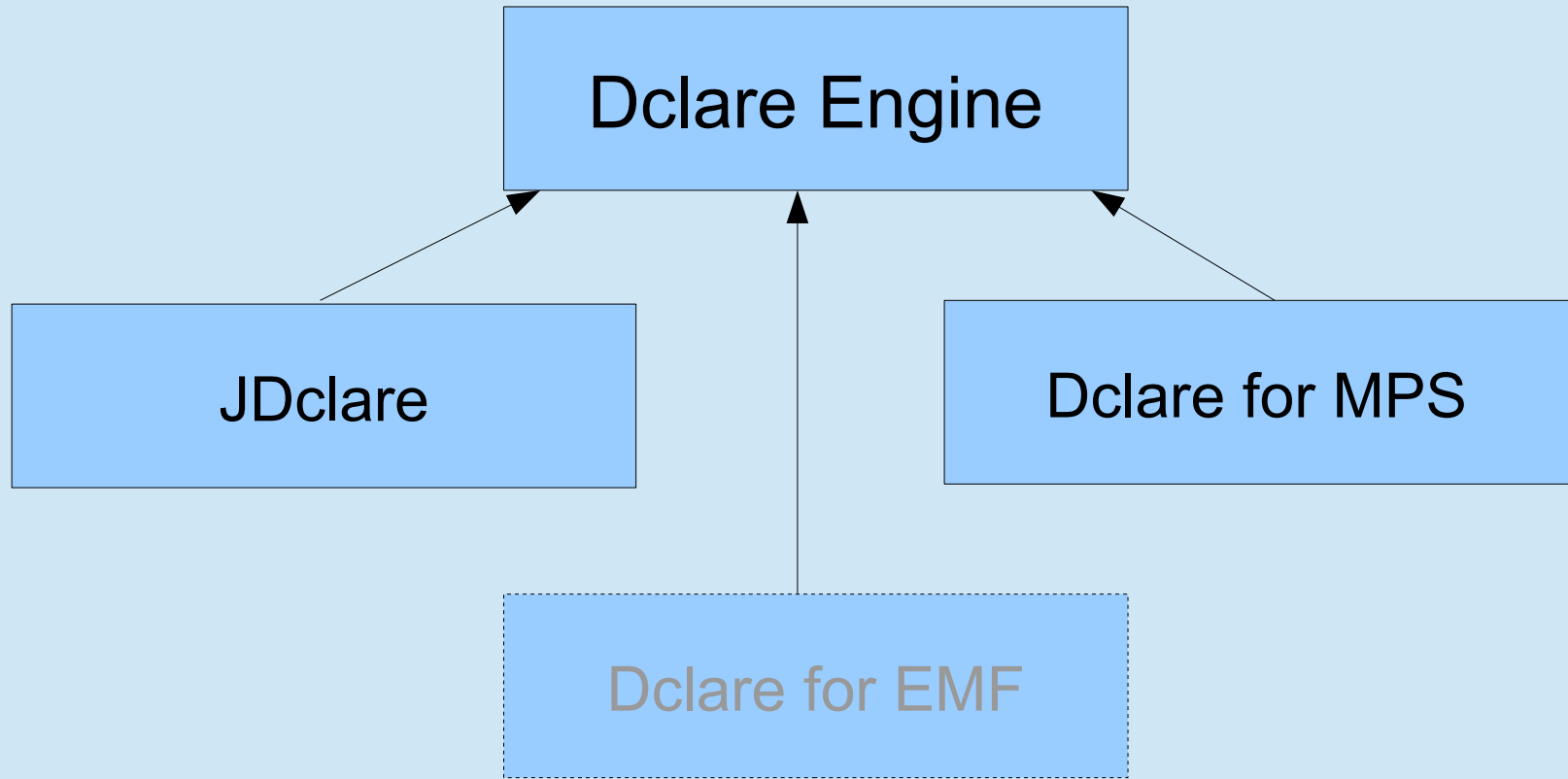
# Dclare for MPS

A declarative base language for MPS

<https://github.com/ModelingValueGroup/DclareForMPS>







all invariants  
hold eventually

hides threading and  
control completely

Rule-based

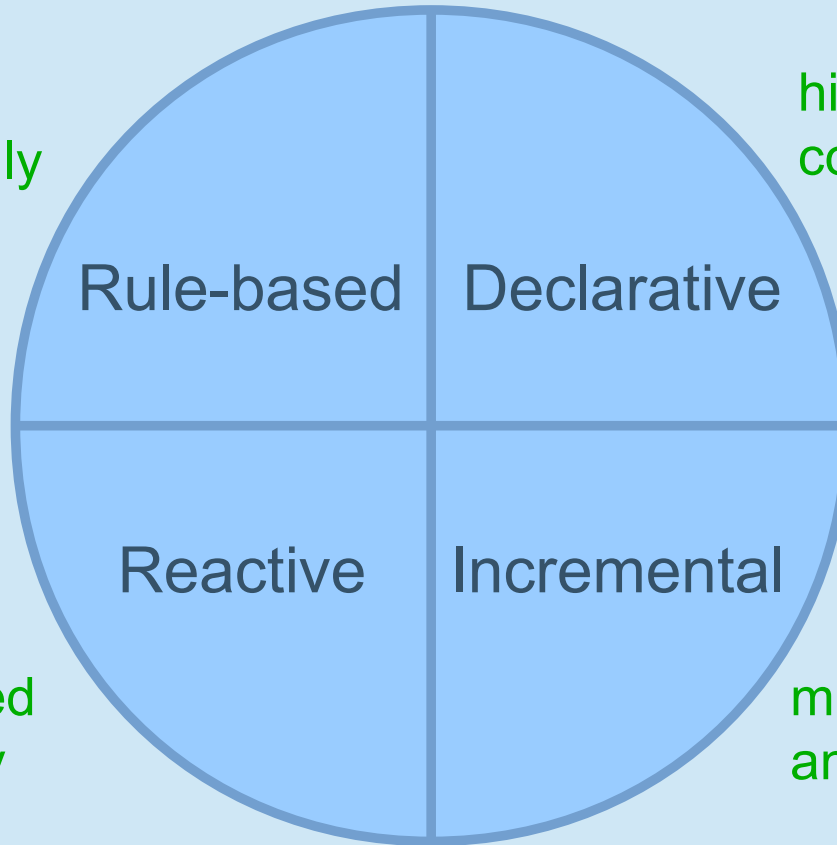
Declarative

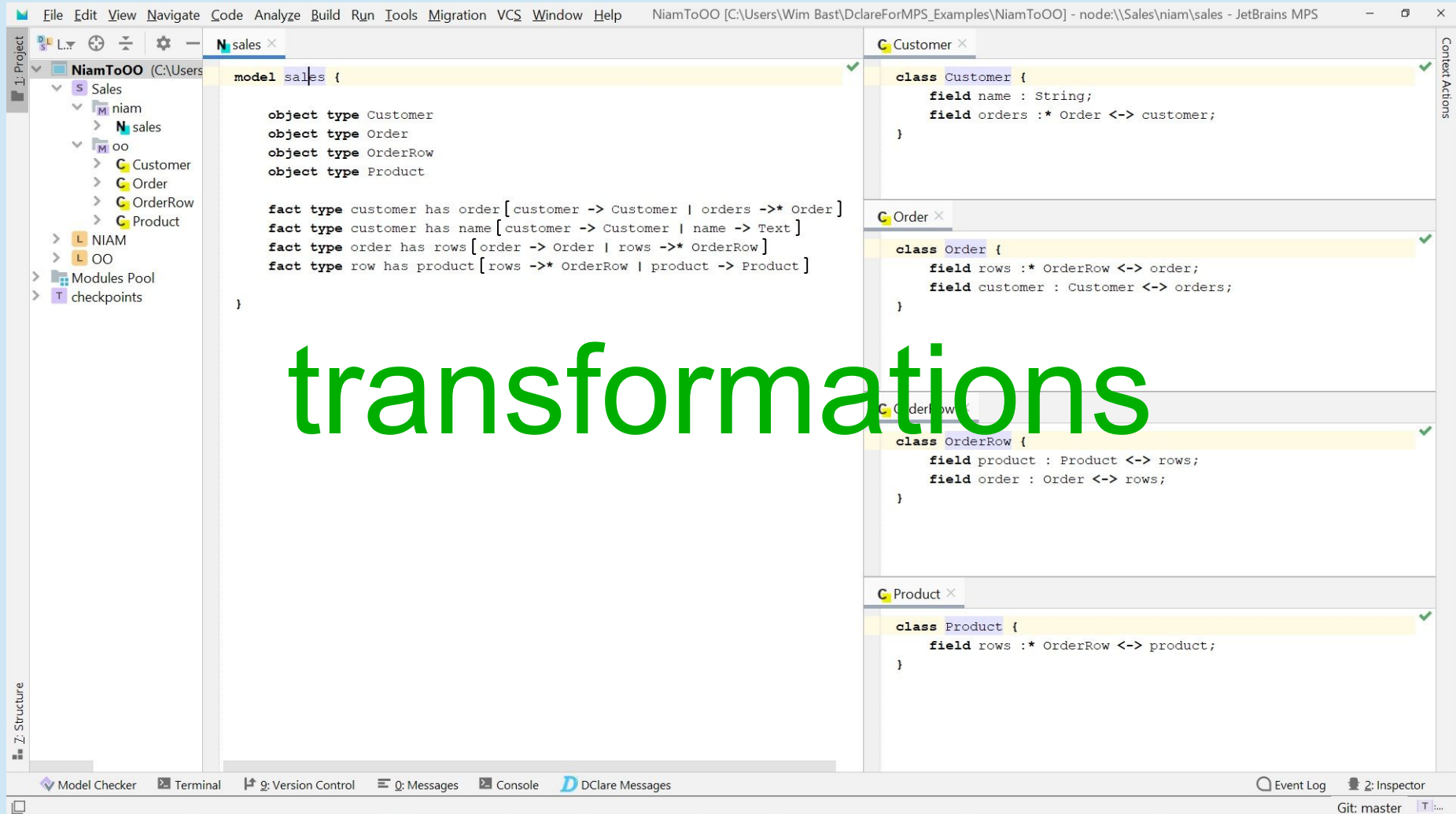
Reactive

Incremental

rules triggered  
automatically

minimal triggering  
and changes





```
model sales {  
  object type Customer  
  object type Order  
  object type OrderRow  
  object type Product  
  
  fact type customer has order [customer -> Customer | orders ->* Order]  
  fact type customer has name [customer -> Customer | name -> Text]  
  fact type order has rows [order -> Order | rows ->* OrderRow]  
  fact type row has product [rows ->* OrderRow | product -> Product]  
}
```

```
class Customer {  
  field name : String;  
  field orders :* Order <-> customer;  
}
```

```
class Order {  
  field rows :* OrderRow <-> order;  
  field customer : Customer <-> orders;  
}
```

```
class OrderRow {  
  field product : Product <-> rows;  
  field order : Order <-> rows;  
}
```

```
class Product {  
  field rows :* OrderRow <-> product;  
}
```

# transformations

```

CellLogic x
struct ruleset CellLogic {
    // attributes
    identifying node<Cell> cell;
    identifying list<int> filling;
    node<Sudoku> sudoku := cell.sudoku;
    list<struct<Context>> contexts := [
        sudoku.rows[cell.row],
        sudoku.columns[cell.column],
        sudoku.blocks[cell.block]];
    list<set<struct<CellLogic>> others := contexts.select({~c =>
        c.cells.where({~c => c.cell != cell && !c.filling == cell.filling}).toSet();});
    set<int> takenByOthers := others.selectMany({~c => c; })
        .where({~c => c.possible.size == 1; })
        .select({~c => c.possible.first; });
    set<int> possible := cell.isSet ? new singleton<int>(cell.value) :
        possible.size != 1 ? sudoku.values.where({~v => !takenByOthers.contains(v); }) : possible;
    list<set<int>> possibleByOthers := others.select({~o => o.possible.toSet; });
    // rules
    if (possible.size > 1) {
        foreach po in possibleByOthers {
            set<int> must = possible.where({~v => !po.contains(v); }).toSet;
            if (must.isNotEmpty) {
                possible := must;
            }
        }
    }
}

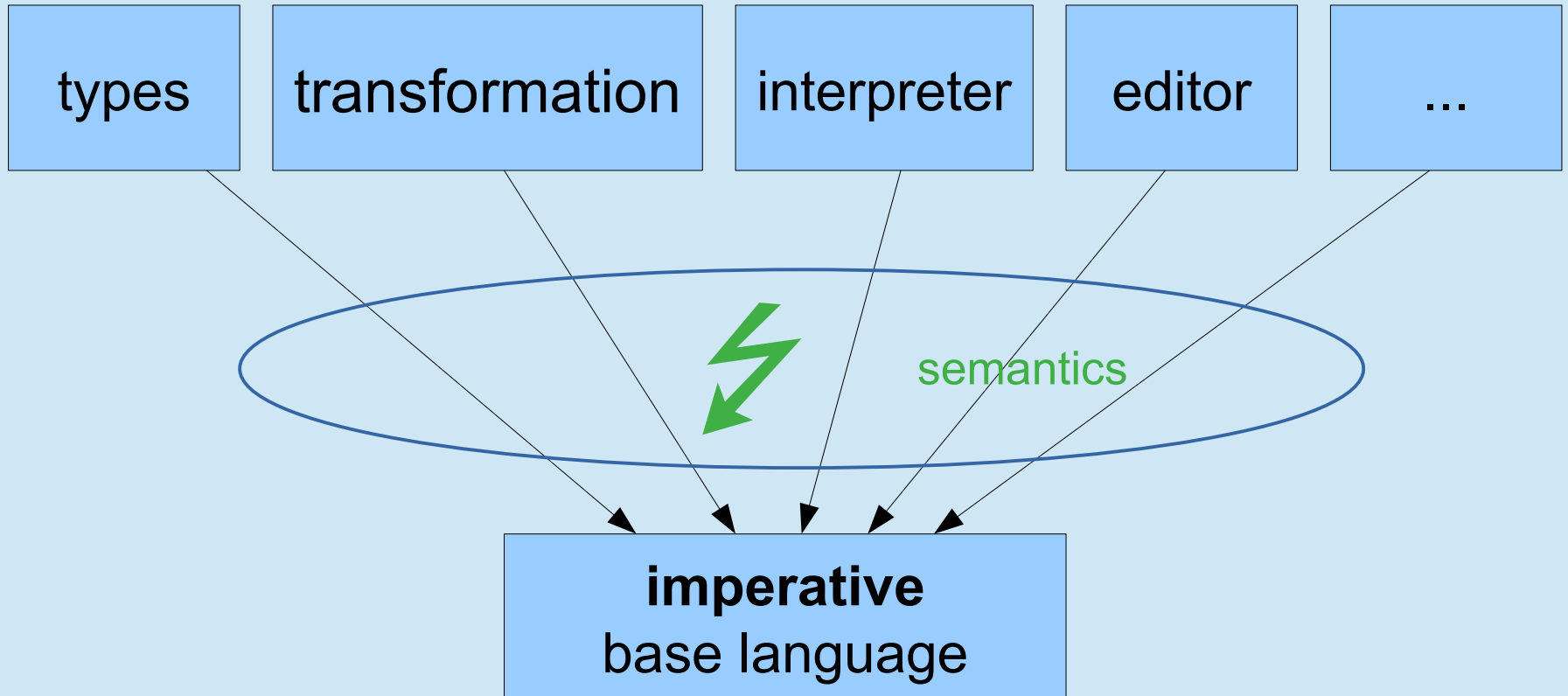
```

sudoku31 x

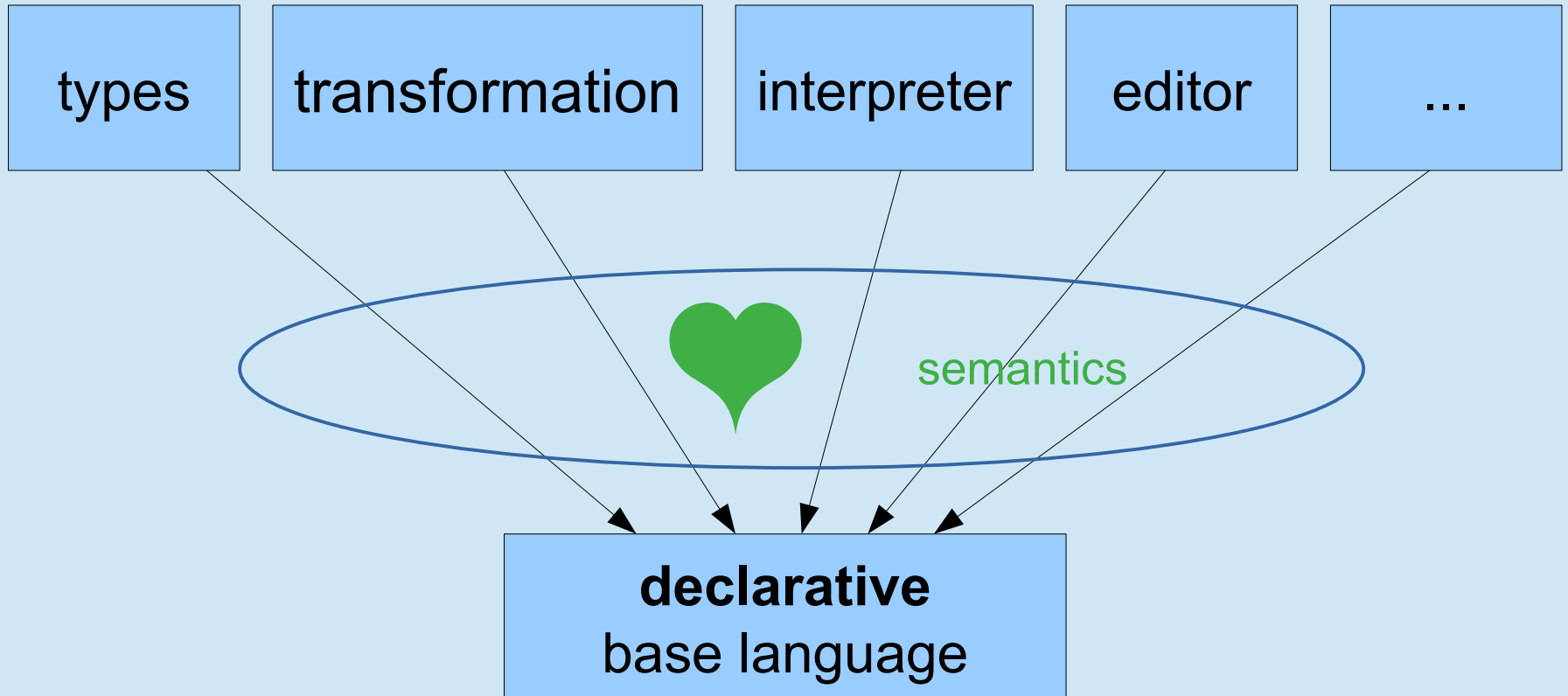
sudoku31 base size 3

5						8		
4			2		8		9	
				7		6		
		2						
	3	5			9			
6		7	5	3				
2		9	1	6	3	4	7	8
		4	8	9				2
		8	4			1		9

interpreters







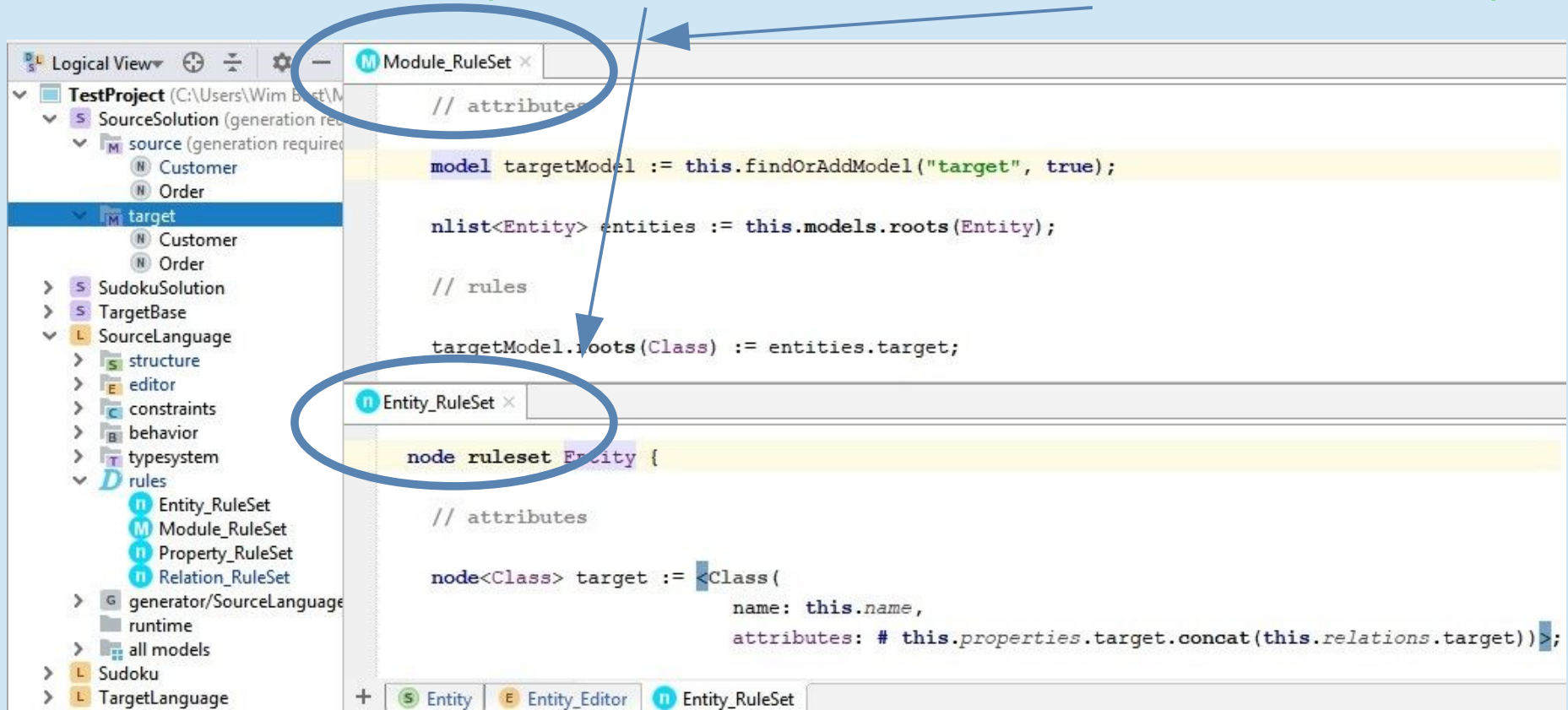
The screenshot shows a modeling tool interface with a logical view on the left and two code editors on the right. The logical view shows a tree structure for a 'TestProject' with various elements like 'SourceSolution', 'source', 'Customer', 'Order', 'target', 'SudokuSolution', 'TargetBase', 'SourceLanguage', 'structure', 'editor', 'constraints', 'behavior', 'typesystem', 'rules', 'generator/SourceLanguage', 'runtime', 'all models', 'Sudoku', and 'TargetLanguage'. The 'rules' folder is circled in blue, and an arrow points from this circle to the text 'complete language aspect' below. The code editors show the following code:

```
// attributes  
model targetModel := this.findOrAddModel("target", true);  
  
nlist<Entity> entities := this.models.roots(Entity);  
  
// rules  
targetModel.roots(Class) := entities.target;
```

```
node ruleset Entity {  
  
// attributes  
  
node<Class> target := <Class(  
    name: this.name,  
    attributes: # this.properties.target.concat(this.relations.target) >;
```

complete language aspect

# rule-sets (on nodes, models, modules, structs)



The screenshot shows a modeling tool interface with a logical view on the left and a code editor on the right. The logical view shows a project structure with a 'target' node selected. The code editor shows the implementation of a 'Module\_RuleSet' and an 'Entity\_RuleSet'.

**Module\_RuleSet** (circled in blue):

```
// attributes
model targetModel := this.findOrAddModel("target", true);

nlist<Entity> entities := this.models.roots(Entity);

// rules
targetModel.roots(Class) := entities.target;
```

**Entity\_RuleSet** (circled in blue):

```
node ruleset Entity {
  // attributes

  node<Class> target := <Class(
    name: this.name,
    attributes: # this.properties.target.concat(this.relations.target)>;
```

Blue arrows point from the green text 'rule-sets (on nodes, models, modules, structs)' to the circled tabs and the corresponding code blocks.

attributes of any type

The screenshot shows a modeling tool interface with a Logical View on the left and a code editor on the right. The Logical View shows a project structure with a 'target' model selected. The code editor displays the following code:

```
// attributes
model targetModel := this.findOrAddModel("target", true);

nlist<Entity> entities := this.models.roots(Entity);

// rules

targetModel.roots(Class) := entities.target;
```

The code editor also shows the definition of the 'Entity' rule set:

```
node ruleset Entity {

// attributes

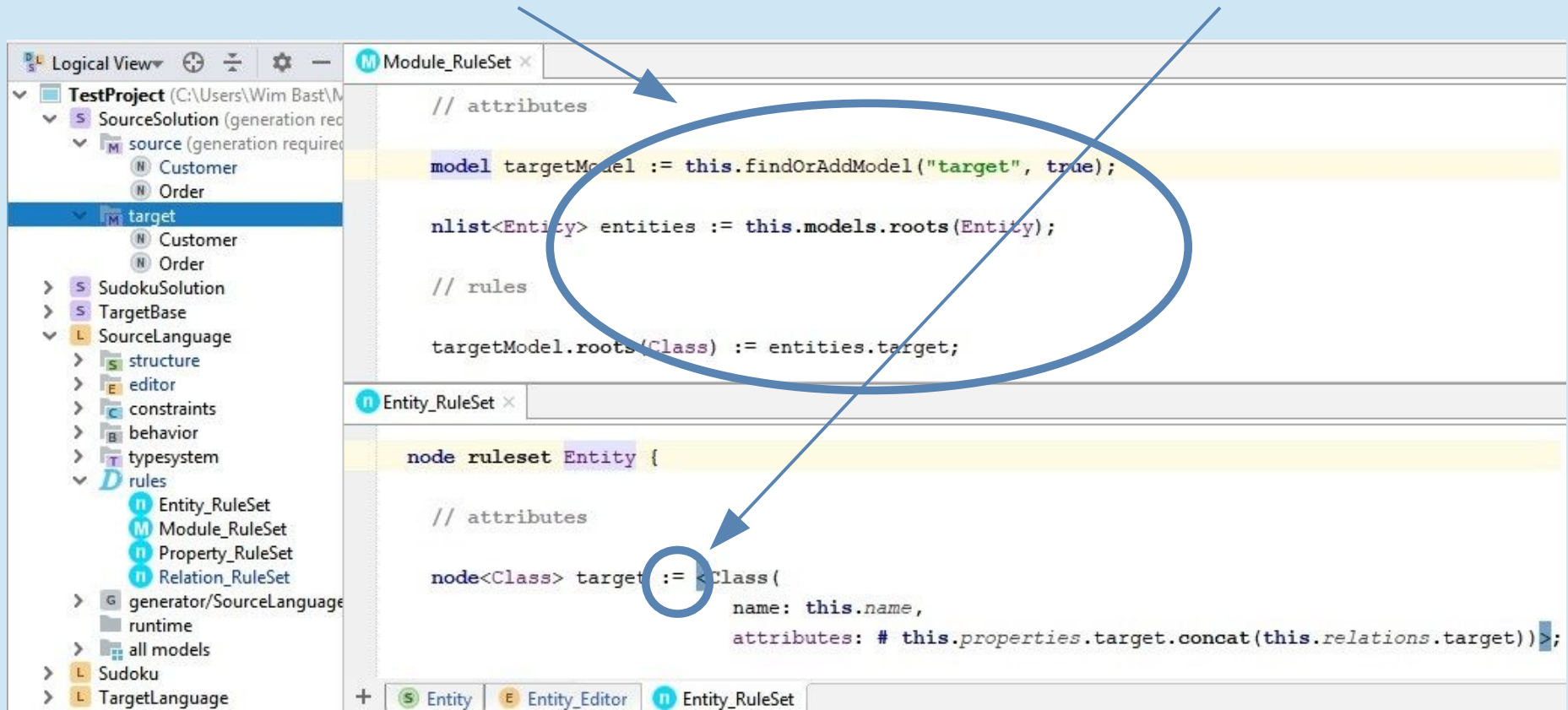
node<Class> target := <Class(
    name: this.name,
    attributes: # this.properties.target.concat(this.relations.target));
```

Two blue circles highlight the following lines of code:

- `model targetModel := this.findOrAddModel("target", true);`
- `node<Class> target := <Class(`

Arrows point from these circles to the text "attributes of any type" above.

# Rules: base-language & equations



The screenshot displays a logical view on the left and two code editors on the right. The top editor, titled 'Module\_RuleSet', contains the following code:

```
// attributes
model targetModel := this.findOrAddModel("target", true);
nlist<Entity> entities := this.models.roots(Entity);

// rules
targetModel.roots(Class) := entities.target;
```

The bottom editor, titled 'Entity\_RuleSet', contains the following code:

```
node ruleset Entity {

// attributes

node<Class> target := <Class(
    name: this.name,
    attributes: # this.properties.target.concat(this.relations.target));
```

Annotations in the image include a blue oval around the 'model targetModel' and 'nlist<Entity> entities' lines in the 'Module\_RuleSet' editor, and a blue circle around the 'node<Class> target := <Class(' line in the 'Entity\_RuleSet' editor. Arrows point from the title 'Rules: base-language & equations' to these specific code elements.

Logical View

- TestProject (C:\Users\Wim Bast\...)
  - SourceSolution (generation required)
    - source (generation required)
      - Customer
      - Order
    - target
      - Customer
      - Order
  - SudokuSolution
  - TargetBase
  - SourceLanguage
    - structure
    - editor
    - constraints
    - behavior
    - typesystem
  - rules
    - Entity\_RuleSet
    - Module\_RuleSet
    - Property\_RuleSet
    - Relation\_RuleSet
  - generator/SourceLanguage
  - runtime
  - all models
  - Sudoku
  - TargetLanguage

```
// attributes  
model targetModel := this.findOrAddModel("target", true);  
  
nlist<Entity> entities := this.models.roots(Entity);  
  
// rules  
  
targetModel.roots(Class) := entities.target;
```

```
node ruleset Entity {  
  
// attributes  
  
node<Class> target := <Class(  
    name: this.name,  
    attributes: # this.properties.target.concat(this.relations.target) >;
```

incremental quotations (light or not) and copy



# DEMO

# Dclare for MPS

## Q & A

<https://github.com/ModelingValueGroup/DclareForMPS>