technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 14 für Software Engineering
Arbeitsgruppe AQUA

Meta Programming System: An Introduction

Till Schallau | September 28, 2020

Contents

Contents

Meta Programming System

The **Meta Programming System** (MPS) [1] is a language workbench to create **Domain Specific Languages** (DSL).

MPS uses/provides:

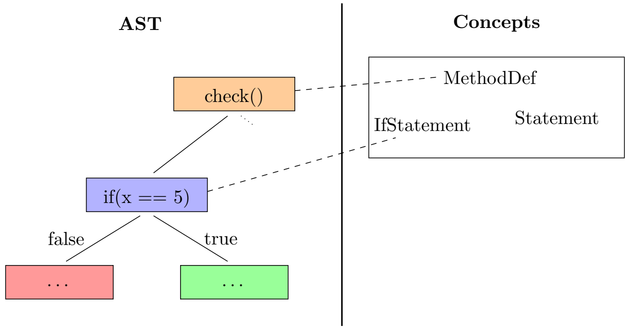
- Code storage in an **Abstract Syntax Tree** (AST)
- Projectional editing
- Code generation
- Language extension possibilities

[1] `https://www.jetbrains.com/mps/`

Abstract Syntax Tree

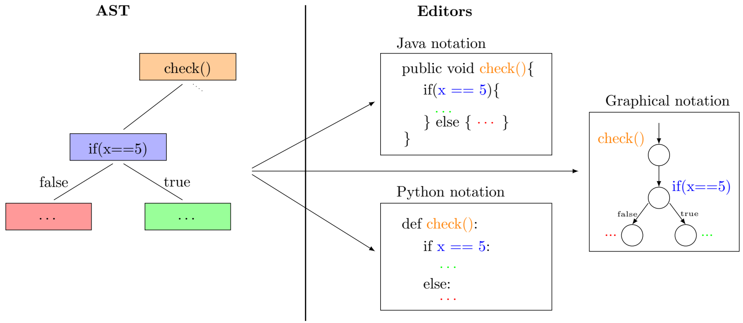
MPS is using an AST as its underlying model, therefore no specific parser is necessary.

The language definition is based on AST-nodes, which build the abstract syntax tree.



Projectional Editing

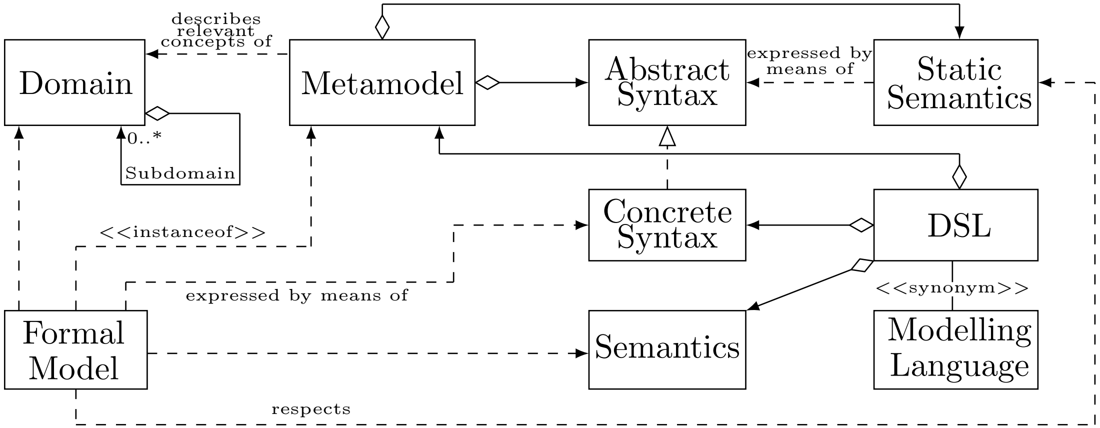
The **Projectional Editor** of MPS is a visual representation of the current AST. It is possible to have multiple editors with different presentation aspects.



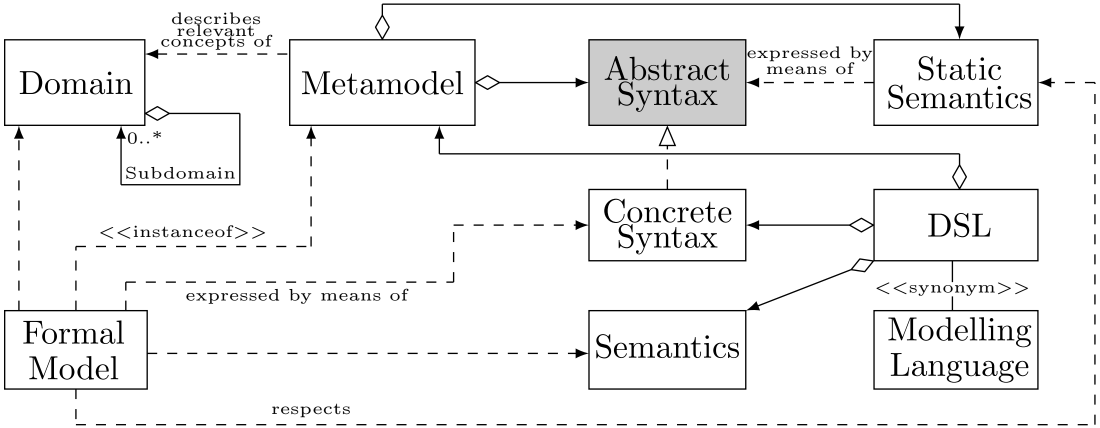
```
public void check(){
    if(x == 5){
        ...
    } else { ... }
}
```

```
def check():
    if x == 5:
        ...
    else:
        ...
```

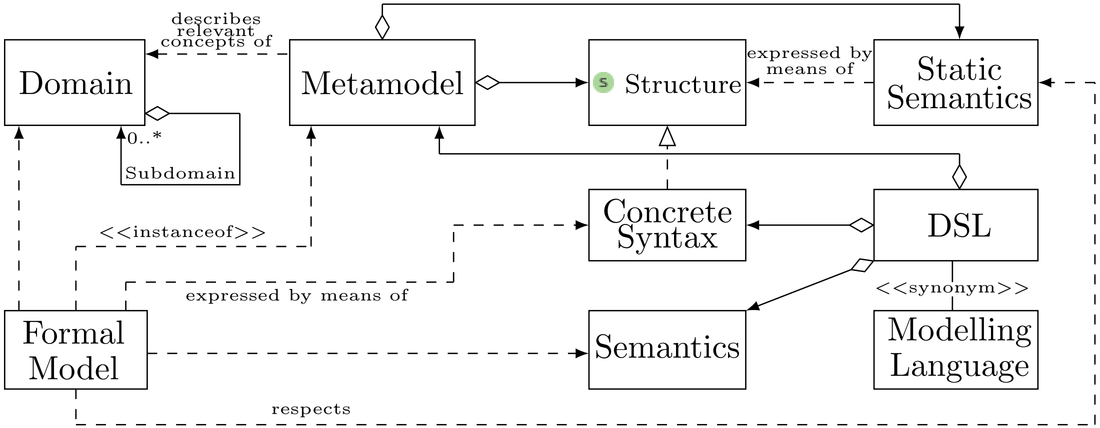
Model-Driven Engineering



Model-Driven Engineering



Model-Driven Engineering



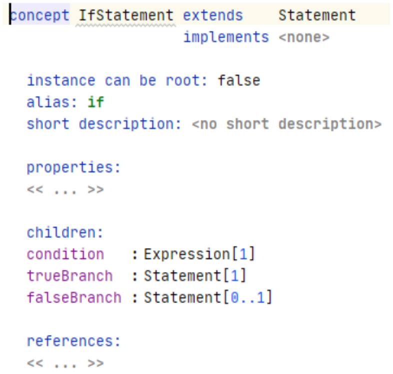
Contents

Structure[2]

The structure of a language is defined as a **Concept** in MPS

Concept:

- Inheritance
- Implementation of Interface
- Properties:
  - Enumeration
  - Primitive Datatype
  - Constrained Datatype
- Children:
  - Any concept
  - Multiplicities ([1], [1..n], [0..n], [0..1])
- References:
  - Reference to another node



```
concept IfStatement extends    Statement
                    implements <none>

  instance can be root: false
  alias: if
  short description: <no short description>

  properties:
  << ... >>

  children:
  condition   : Expression[1]
  trueBranch  : Statement[1]
  falseBranch : Statement[0..1]

  references:
  << ... >>
```

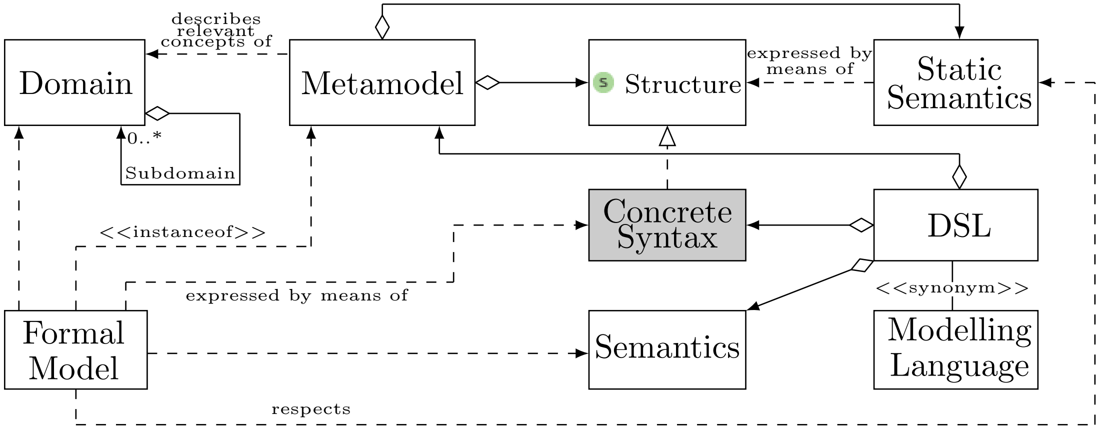
[2] https://www.jetbrains.com/help/mps/structure.html

Hands-On

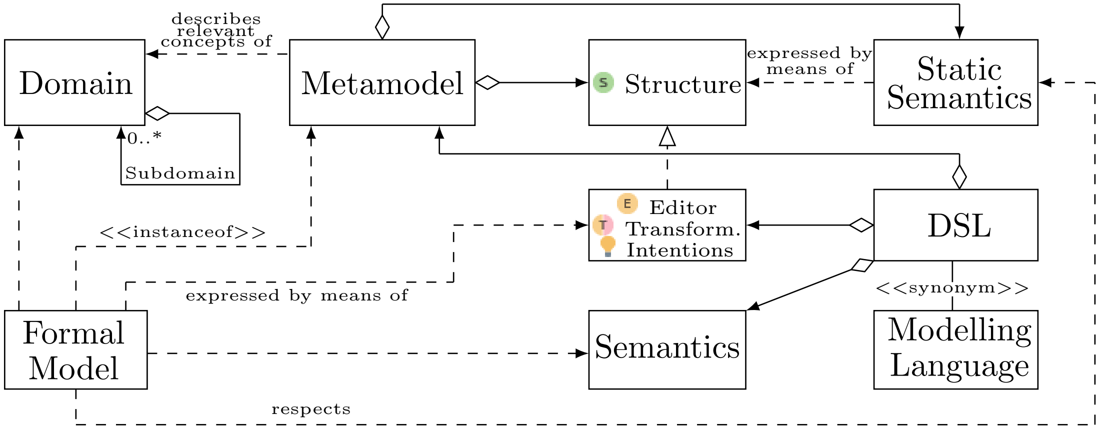
After this introduction into the structure of MPS, there is a repository with all necessary information under `https://github.com/tillschallau/mps-workshop`

Contents

Model-Driven Engineering



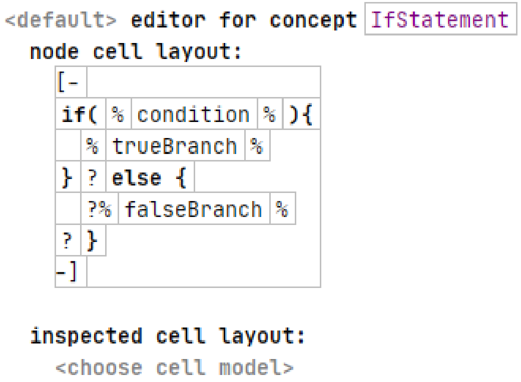
Model-Driven Engineering



Editor[3]

Types of Cell Models:

- **Constant cell**: `<constant>`
- **Property cell**: `{property}`
- **Child cell**: `%child%`
- **Referent cell**: `(%reference%->{name})`
- **Child list cell**: `(>%child%/empty cell: <default><)`
- **Model access**: `*model access*`
- **Collection cell**: `[- -]` (indent layout) or
  `[> <]` (horizontal) or
  `[/ /]` (vertical)



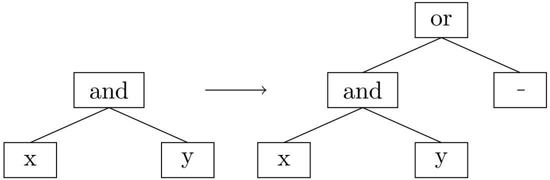
```
<default> editor for concept IfStatement
  node cell layout:
    [-
    if( % condition % ){
      % trueBranch %
    } ? else {
      ?% falseBranch %
    ? }
    -]

  inspected cell layout:
    <choose cell model>
```

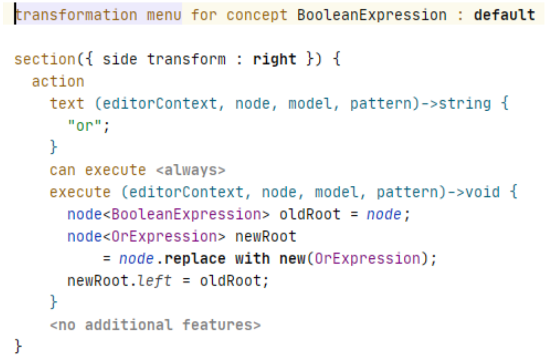
[3] https://www.jetbrains.com/help/mps/editor.html

Transformations[4]

Transformations let you edit the AST by replacing and moving AST nodes

Example:
`(x and y)` type `or` should yield
`((x and y) or _)`





```
transformation menu for concept BooleanExpression : default

section({ side transform : right }) {
  action
    text (editorContext, node, model, pattern)->string {
      "or";
    }
    can execute <always>
    execute (editorContext, node, model, pattern)->void {
      node<BooleanExpression> oldRoot = node;
      node<OrExpression> newRoot
          = node.replace with new(OrExpression);
      newRoot.left = oldRoot;
    }
    <no additional features>
}
```

[4]https://www.jetbrains.com/help/mps/transformation-menu-language.html

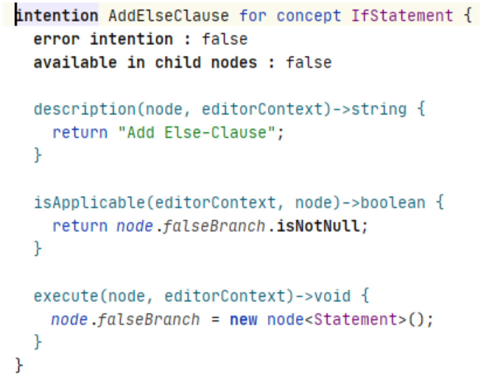
Intentions[5]

Intentions:

- Provide Intention menu by pressing Alt + ↲
- Execute predefined actions
- Can be used to correct errors (**error intention**)

Variants:

- Intention
- Universal Intention
- Surround With Intention
- Parameterized Intention

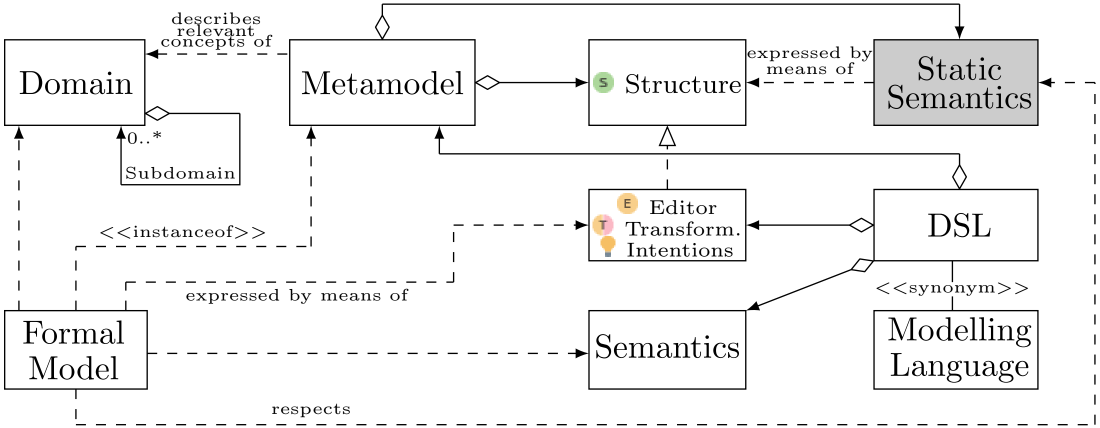


```
intention AddElseClause for concept IfStatement {
  error intention : false
  available in child nodes : false

  description(node, editorContext)->string {
    return "Add Else-Clause";
  }

  isApplicable(editorContext, node)->boolean {
    return node.falseBranch.isNotNull;
  }

  execute(node, editorContext)->void {
    node.falseBranch = new node<Statement>();
  }
}
```

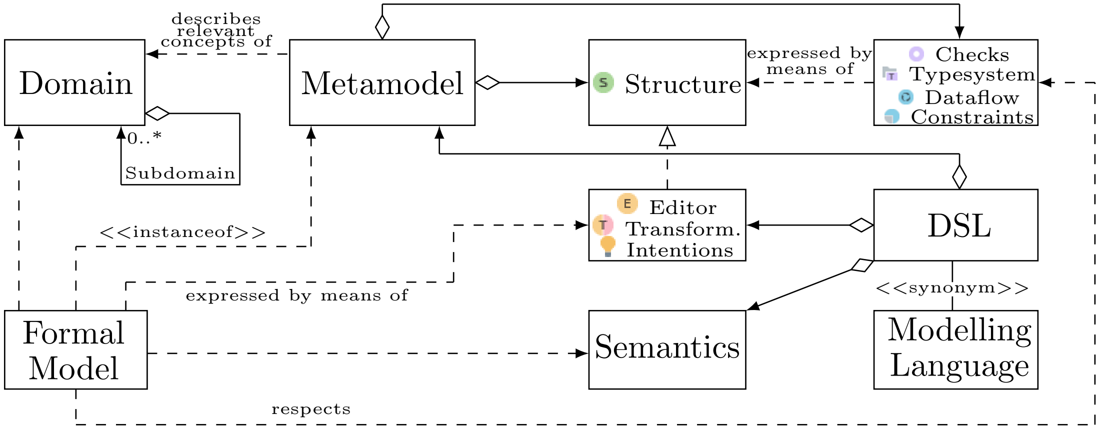
[5] https://www.jetbrains.com/help/mps/mps-intentions.html

Contents

Model-Driven Engineering



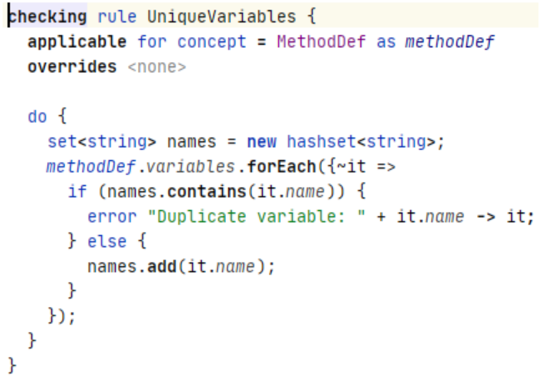
Model-Driven Engineering



Checking Rules[6]

Checks:

- Inspect the model for known error patterns
- Static code analysis
- Reports found errors/warnings/infos
- Can provide quick fixes for errors/warnings

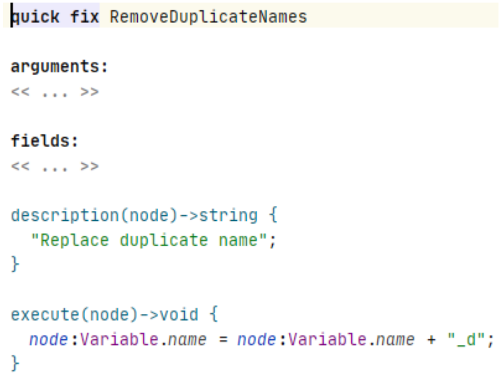


```
checking rule UniqueVariables {
  applicable for concept = MethodDef as methodDef
  overrides <none>

  do {
    set<string> names = new hashset<string>;
    methodDef.variables.forEach({~it =>
      if (names.contains(it.name)) {
        error "Duplicate variable: " + it.name -> it;
      } else {
        names.add(it.name);
      }
    });
  }
}
```

[6] https://www.jetbrains.com/help/mps/typesystem.html

Quick Fixes[7]

Quick Fixes can be attached to the `error`, `warning` or `info` call in a checking rule via the Inspector menu

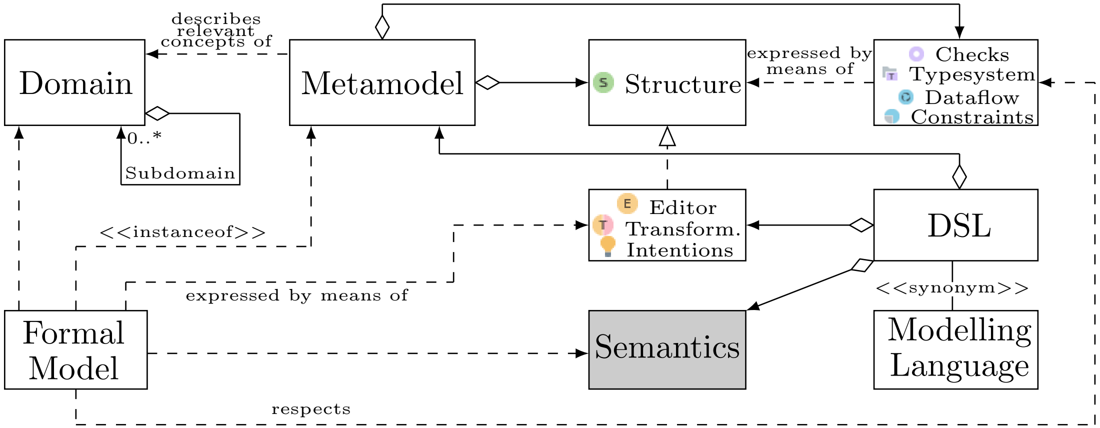


```
quick fix RemoveDuplicateNames

arguments:
<< ... >>

fields:
<< ... >>

description(node)->string {
  "Replace duplicate name";
}

execute(node)->void {
  node:Variable.name = node:Variable.name + "_d";
}
```

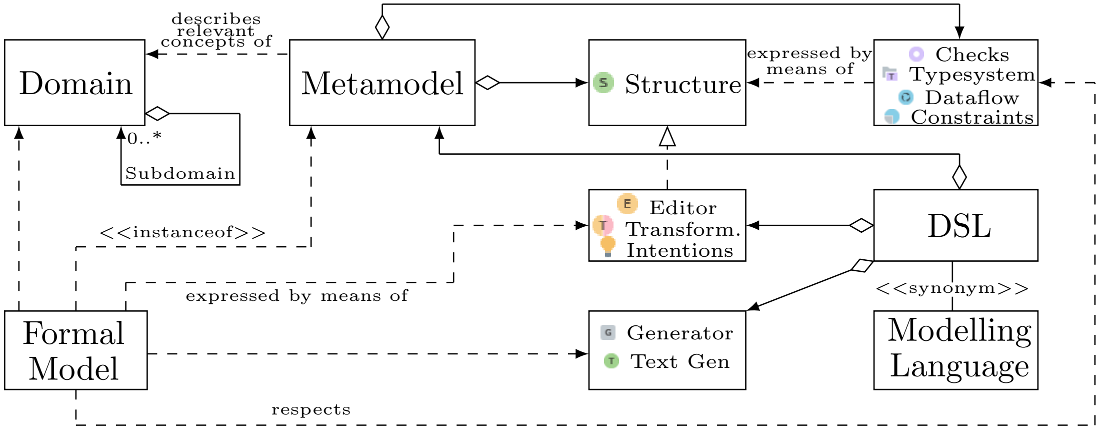
[7] https://www.jetbrains.com/help/mps/typesystem.html

Contents

Model-Driven Engineering



Model-Driven Engineering



Language Generation

In MPS there are two possible ways of generating (here: transforming) code.

Model-To-Model Transformation

- Translate models into other models
- Predefined models exist:
  - Base Language (Java)
  - mbeddr (C and C extensions) [8]
  - MPS CSharp (C#) [9]
- Does not flush text into file
- Each predefined model has an associated Model-To-Text Transformation

$\Longrightarrow$

Model-To-Text Transformation

- Convert a model into text
- Give the output some reasonable layout
- Lets you define a file ending (e.g. `.java`)
- Flush text into a file

[8] `http://mbeddr.com/`

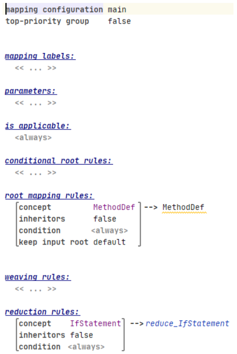
[9] `https://github.com/vaclav/MPS_CSharp`

Model-To-Model Transformation [10]

Each generator consists of **Mapping Configurations** that combines all templates

Some Generator Rules:

- **Root Mapping Rule:** Generates a root node in the output model
- **Reduction Rule:** Transforms a node based on a template
- **Mapping Label:** Helper for name consistency throughout generation



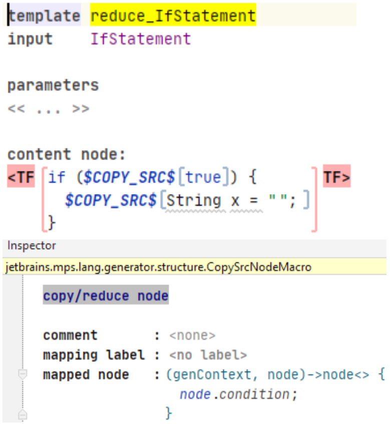
```
mapping configuration main
top-priority group    false


mapping labels:
  << ... >>

parameters:
  << ... >>

is applicable:
  <always>

conditional root rules:
  << ... >>

root mapping rules:
  [concept          MethodDef ] --> MethodDef
  [inheritors       false     ]
  [condition       <always>   ]
  [keep input root default    ]


weaving rules:
  << ... >>

reduction rules:
  [concept    IfStatement] --> reduce_IfStatement
  [inheritors false      ]
  [condition  <always>   ]
```

[10] https://www.jetbrains.com/help/mps/generator-language.html

Templates [11]

Template Macros used in a **Template Fragment <TF TF>**:

- **Property $[]:** Computes value of a property
- **Reference ->$[]:** Computes referent node
- **IF[]:** Conditional generation of template code
- **$LOOP$[]:** Applies template to set of nodes
- **$CALL$[]:** Calls another template with parameters
- **$COPY_SRC$[]:** Copies node
- **$LABEL$[]:** Registers generated name into generation context



```
template reduce_IfStatement
input    IfStatement

parameters
<< ... >>

content node:
<TF [if ($COPY_SRC$[true]) {          ] TF>
      $COPY_SRC$[String x = " "; ]
    }
```

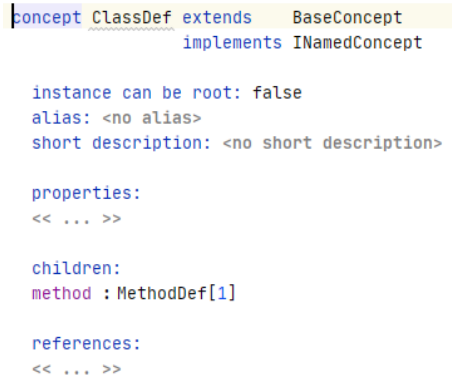
Inspector
jetbrains.mps.lang.generator.structure.CopySrcNodeMacro

```
copy/reduce node

comment       : <none>
mapping label : <no label>
mapped node   : (genContext, node)->node<> {
                  node.condition;
                }
```

[11]https://www.jetbrains.com/help/mps/generator-language.html

Template Combination 1/4

Now the `MethodDef` concept is contained in a `ClassDef`

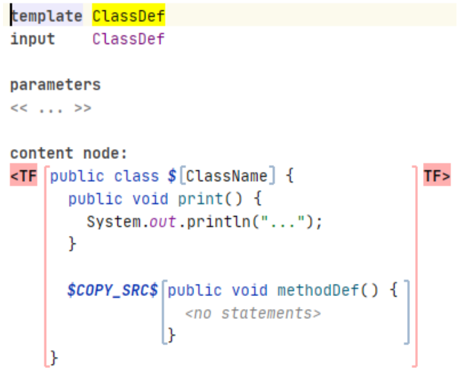


```
concept ClassDef extends    BaseConcept
                 implements INamedConcept

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  method : MethodDef[1]

  references:
  << ... >>
```

Template Combination 2/4

The ClassDef generator template contains a statically generated method print()

The ClassDef generator template contains the generated output of the MethodDef concept



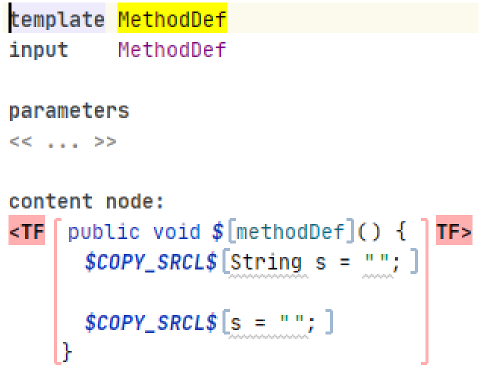
```
template ClassDef
input    ClassDef

parameters
<< ... >>

content node:
<TF [public class $[ClassName] {                  ] TF>
      public void print() {
        System.out.println("...");
      }

      $COPY_SRC$[public void methodDef() { ]
                  <no statements>
                }
    }
```

Template Combination 3/4

To generate the method, generate its children:

- `Variables` (e.g. `$COPY_SRCL$[String s = "";]`)
- `Statements` (e.g. `$COPY_SRCL$[s = "";]`)

What is now needed to always call the `print`-method of the `ClassDef`-concept generation output? The method is currently not available in this context.

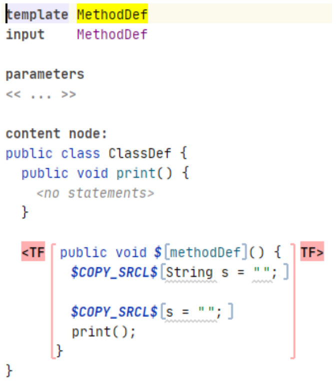


```
template MethodDef
input    MethodDef

parameters
<< ... >>

content node:
<TF [ public void $[methodDef]() {   ] TF>
      $COPY_SRCL$[String s = ""; ]

      $COPY_SRCL$[s = ""; ]
    }
```

Template Combination 4/4

To use the print-method, simulate the surrounding environment necessary for the generation (e.g. the surrounding class with its print()-method)

Mark the code that should be generated with the Template Fragment macro

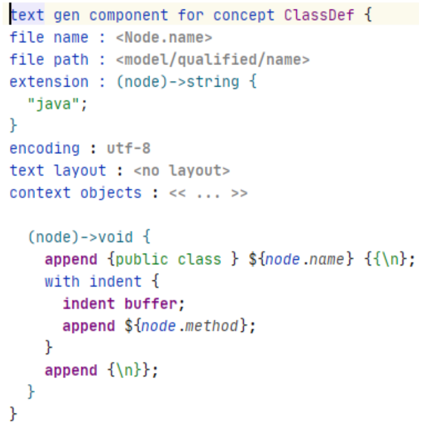


```
template MethodDef
input    MethodDef

parameters
<< ... >>

content node:
public class ClassDef {
  public void print() {
    <no statements>
  }

  <TF [ public void $[methodDef]() { ] TF>
          $COPY_SRCL$[String s = ""; ]

          $COPY_SRCL$[s = ""; ]
          print();
        }
}
```

TextGen [8]

The TextGen language operations:

- **append:** append text of the following kind:
  - **{string value}:** constant text
  - **\n:** line break
  - **$list{node.list}:** list without separator
  - **$list{node.list with ,}:** list with separator ","
  - **${node.child}**
- **with indent { code }:** increase indentation level for code
- **indent buffer:** apply indentation for current line
- **increase depth:** increase indentation level
- **decrease depth:** decrease indentation level



```
text gen component for concept ClassDef {
file name : <Node.name>
file path : <model/qualified/name>
extension : (node)->string {
  "java";
}
encoding : utf-8
text layout : <no layout>
context objects : << ... >>

  (node)->void {
    append {public class } ${node.name} {{\n};
    with indent {
      indent buffer;
      append ${node.method};
    }
    append {\n}};
  }
}
```

[8] https://www.jetbrains.com/help/mps/textgen.html