SE SOFTWARE ENGINEERING

RWTH AACHEN UNIVERSITY

Adaptable Symbol Table Management by Meta Modeling and Generation of Symbol Table Infrastructures

27th October 2015

Katrin Hölldobler, Pedram Mir Seyed Nazari, and Bernhard Rumpe

Software Engineering
RWTH Aachen University

http://www.se-rwth.de/

Motivation

- Many textual software languages share common concepts
  - Define model elements
  - Refer to model elements defined in the same model as well as in another model (including loading of models)
  - Shadow names that are already defined
- Mechanisms behind those concepts usually are complex and must be fully understood by language engineer in order to apply them
- Therefore, language workbenches provide mechanisms to implement those concepts
- The MontiCore language workbench uses so-called symbol tables

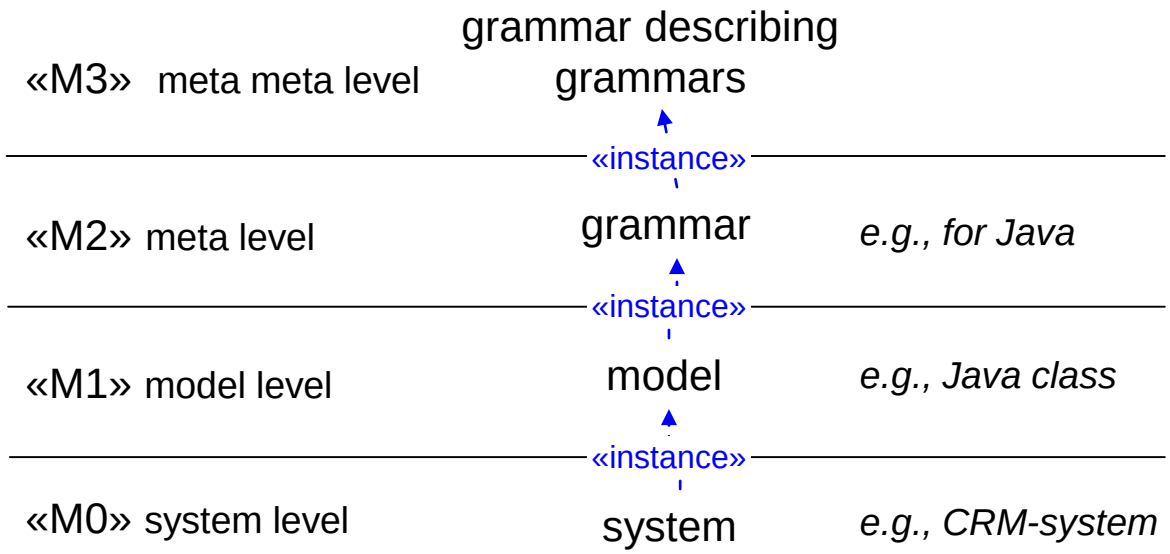
Symbol Tables

- A symbol table is a data structure that maps names to their associated information.

- In MontiCore, a symbol table may also represent the semantic meta model and contain information not directly defined in the model
  - e.g., all non-private fields of a Java class including fields of the super class

Contribution

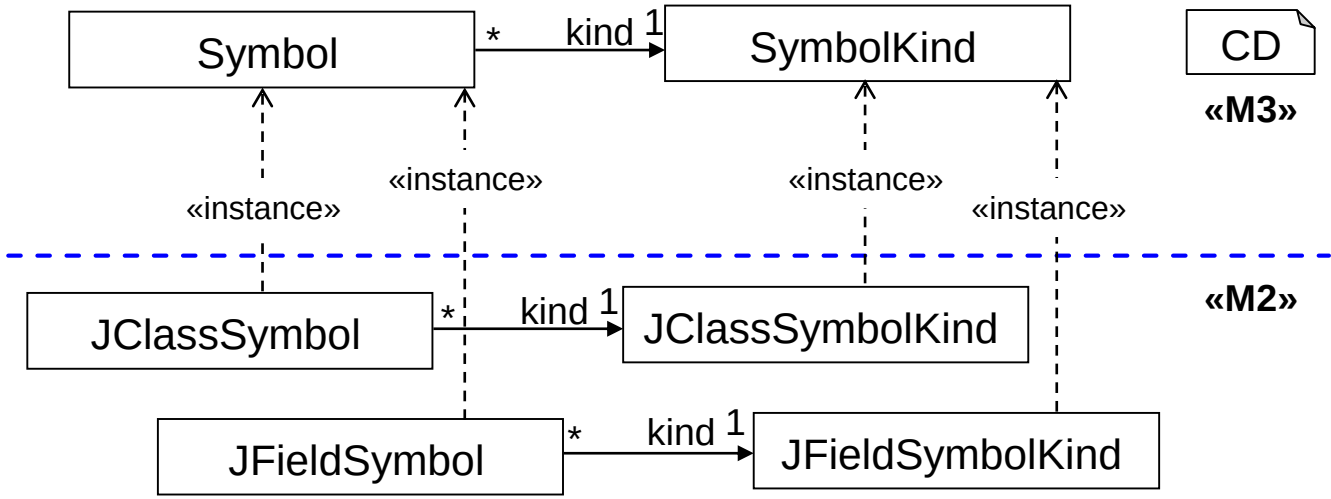
- Language-independent meta model (M3) for symbol tables which is basis for language-specific symbol tables (M2)
- An integration of the symbol table M3 model and the grammar M3 model, which allows to switch between both models as needed
- The generation of the language specific symbol table and automatically integration with the grammar model



Symbols: Named Model Elements

```
class C {
  int f;
  C c;

  void m() {
    int f = g;

    while(…) {
      int f;
    }
  }
}
```

named elements



- A symbol represents a named model element and its associated information
- It may provide information that is not (directly) contained in the model element
  - e.g., all non-private methods of the super class

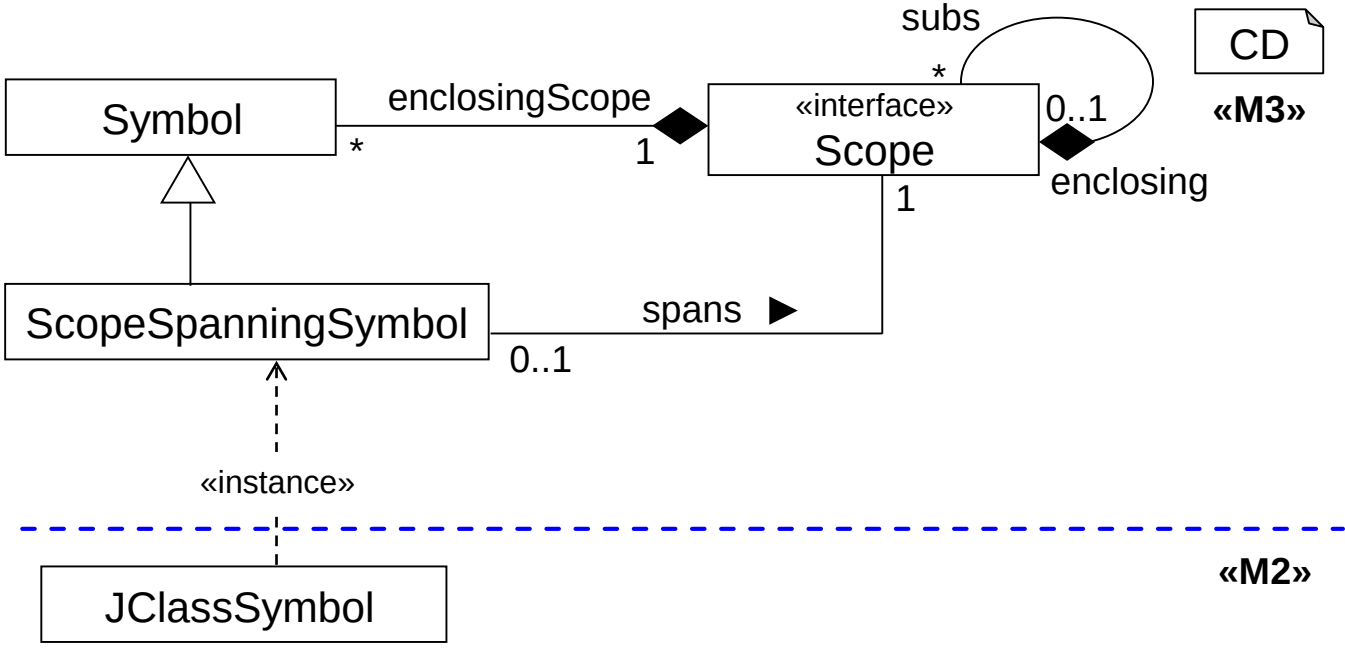
Scopes: Containers for Symbols

```
class C {
   int f;
   C c;

   void m() {
      int f = g;

      while(...) {
         int f;
      }
   }
}
```

if scope

method scope

class scope



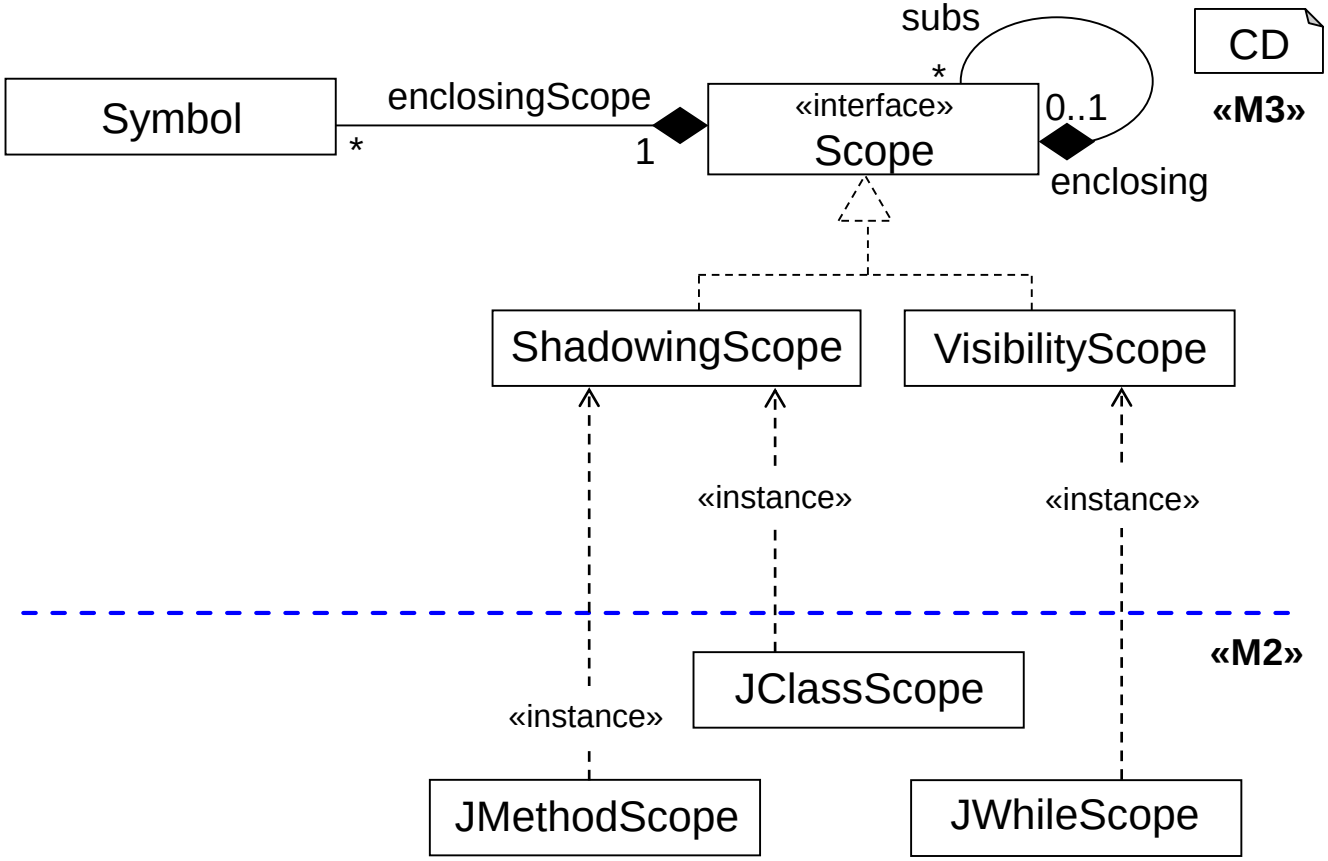
- A scope holds a collection of symbol definitions
- Structured hierarchically
- Limits visibility of a symbol
- Some symbols span a scope (scope spanning symbols)

Shadowing and Visibility Scopes

```
class C {
  int f;
  C c;

  void m() {
    int f;

    while(...) {
      int f;
    }
  }
}
```

shadows field f

not allowed, local variable already defined



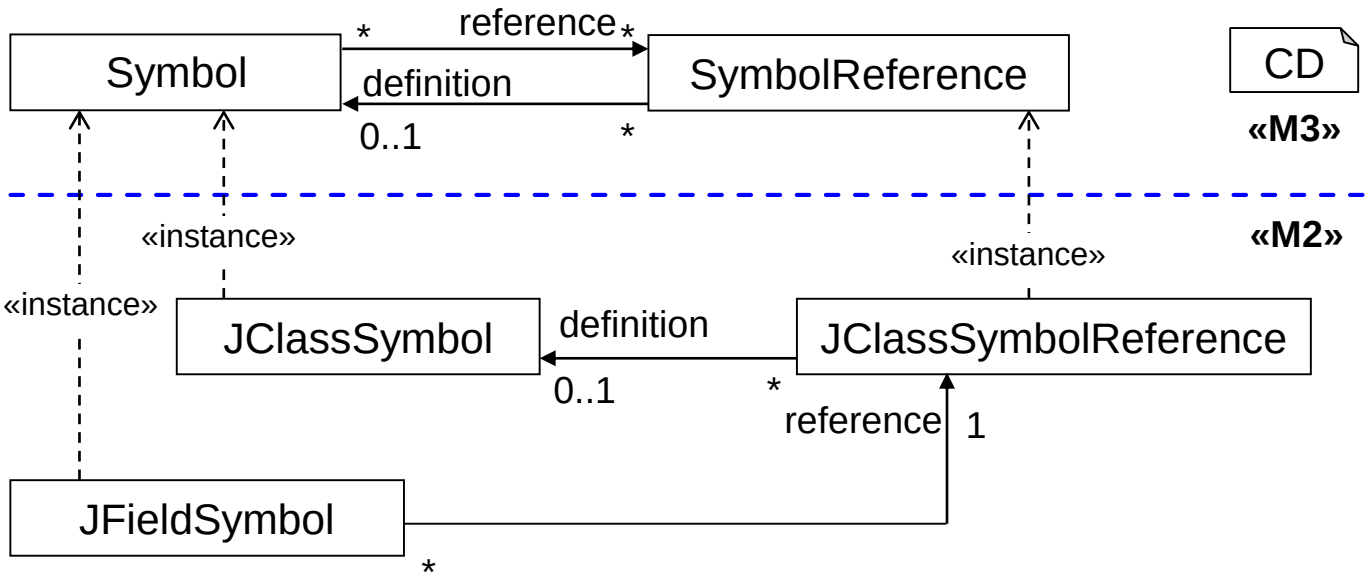
- Shadowing scopes may shadow names of enclosing scopes, visibility scopes may not

Symbol References

```
class C {
   int f;
   C c;
   void m() {
      int f;
      while(...) {
         int f;
      }
   }
}
```

refers to class "C"

refers to built-in type "int"



- A symbol reference refers to a symbol defined elsewhere either in the same model or another

Simplified Grammar M3 Model

- Textual software languages are described by grammars
- Abstract syntax tree is the meta-model

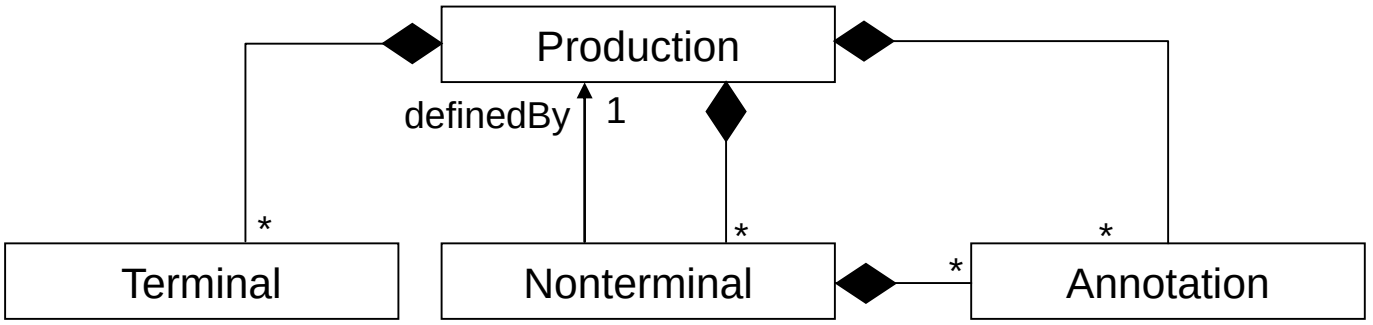
(instance of)
Annotation

```
JClass@Ann = "class" Name "{" (JField | JMethod)* "}";
JField = type:Name Name ";" ;
```

(instance of)
Production

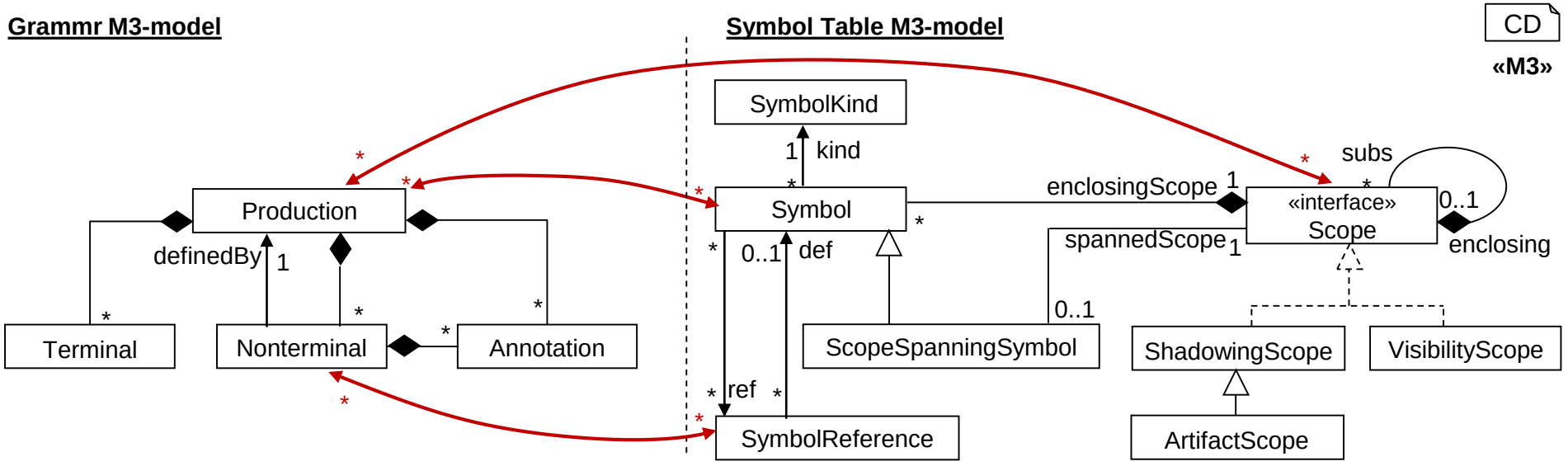
(instance of)
Terminal

(instance of)
Nonterminal



Composing Grammar and Symbol Table M3 Models

- Language engineer (LE) usually needs both M2 models
- To enable this, we compose the M3 models
- LE can switch between these structures as needed



Generating Language-Specific Symbol Table (M2)

- Language-specific symbol table depends on the language's semantic

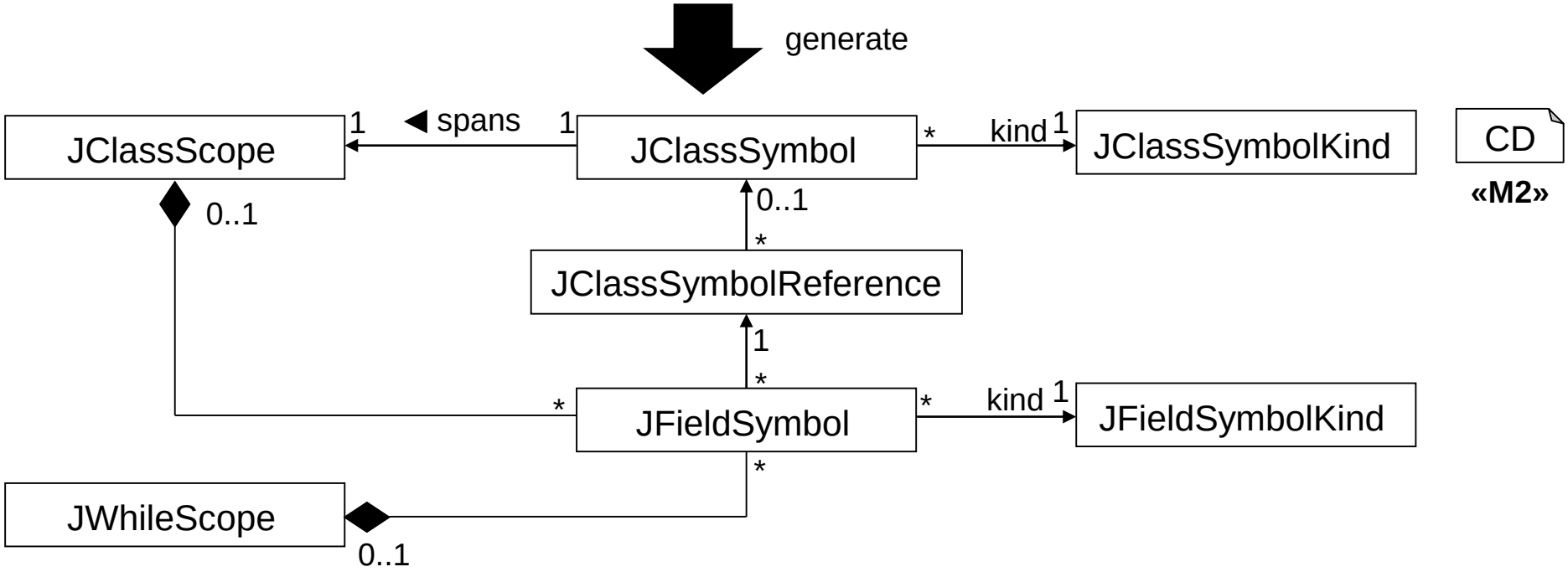
- Composition of the two M2 models is affected both the grammar design as well as the symbol table design
- Hence, composition must be conducted manually

- Generative support
  - Prerequisite: limit cardinalities to 0..1
  - Automatically derive the language-specific symbol table infrastructure (or parts of it) from the grammar
  - Simultaneously integrate it with language-specific grammar model
  - Using annotation mechanism of MontiCore's grammar

Generating Language-Specific Symbol Table

- Mapping via naming convention
  - production **`Prod`** is mapped to symbol **`Prod`*`Symbol`***

```
JClass@! = "class" Name "{" (JField | JMethod)* "}";
JField@! = type:Name@JClass Name ";" ;
JWhile   = "while" "(" ... ")" "{" JField* ... "}";
```



Conclusion

- Textual software languages share some common concepts, such as defining and referencing model elements, and name shadowing
- Language-independent meta model for symbol tables first-level classes, which serves as basis for language-specific symbol tables
- Integration of this the symbol table meta model and the grammar meta model
- Generating language-specific symbol table infrastructure (or parts of it) and directly integrating it with the corresponding grammar model