Towards an Overture Code Generator

Peter Jørgensen Peter Gorm Larsen

Aarhus University, Department of Engineering

28 August 2013
The 11th Overture Workshop
Methods, Tools and Techniques for Modelling in VDM

Outline

Generating code from a model

1. Formal modeling helps understanding the system
2. Reducing the effort for transitioning to implementation
3. Costs of generating code
4. The code generator presented here is early work
   - It generates Java code from VDM++ models

Related work

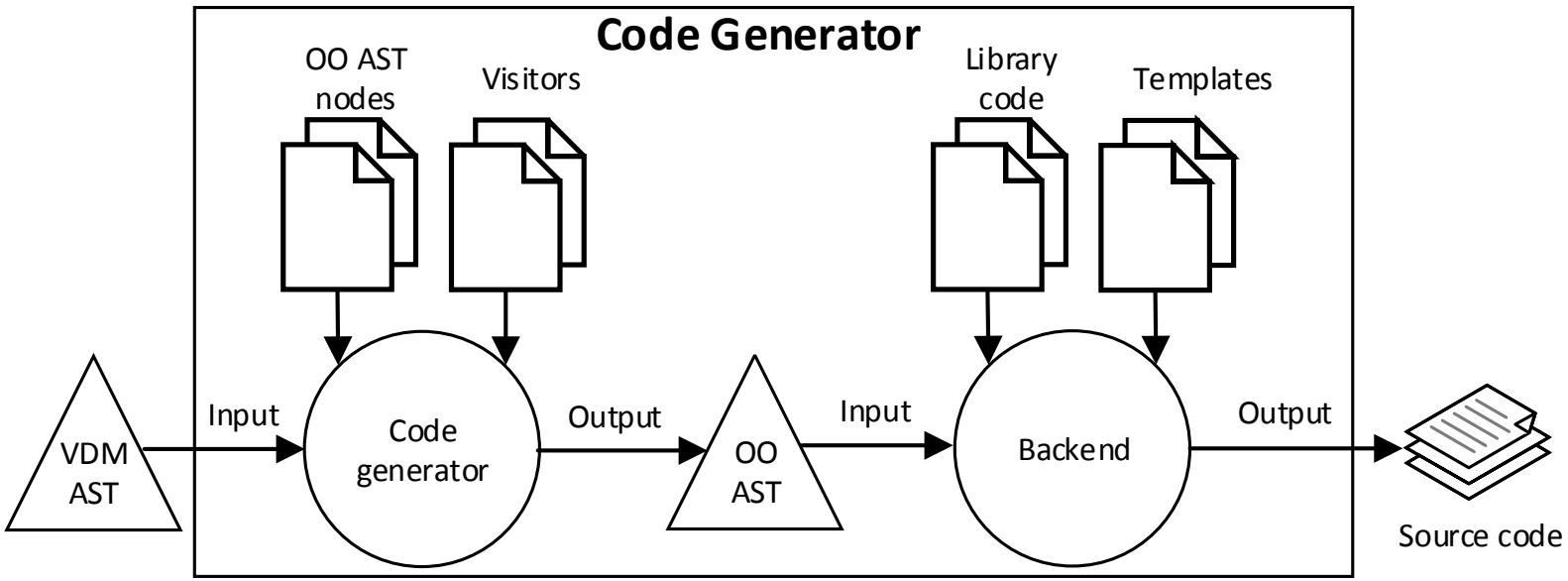
1. VDMTools CG development started in the 1990s
   - First CG support for Java was developed
   - Later support for VDM++ concurrency was made
2. An attempt was made inside the Overture project
   - It was not possible to extract type information
   - It did only work for trivial examples

CG and the new Abstract Syntax Tree (AST) (1/2)

1. The CG is implemented using visitors
2. The visitors construct a new intermediate *OO AST*
3. The OO AST serves two primary purposes
   - It gradually introduces the complexity of CG
   - It enables extending the CG with multiple OO languages

CG and the new AST (2/2)

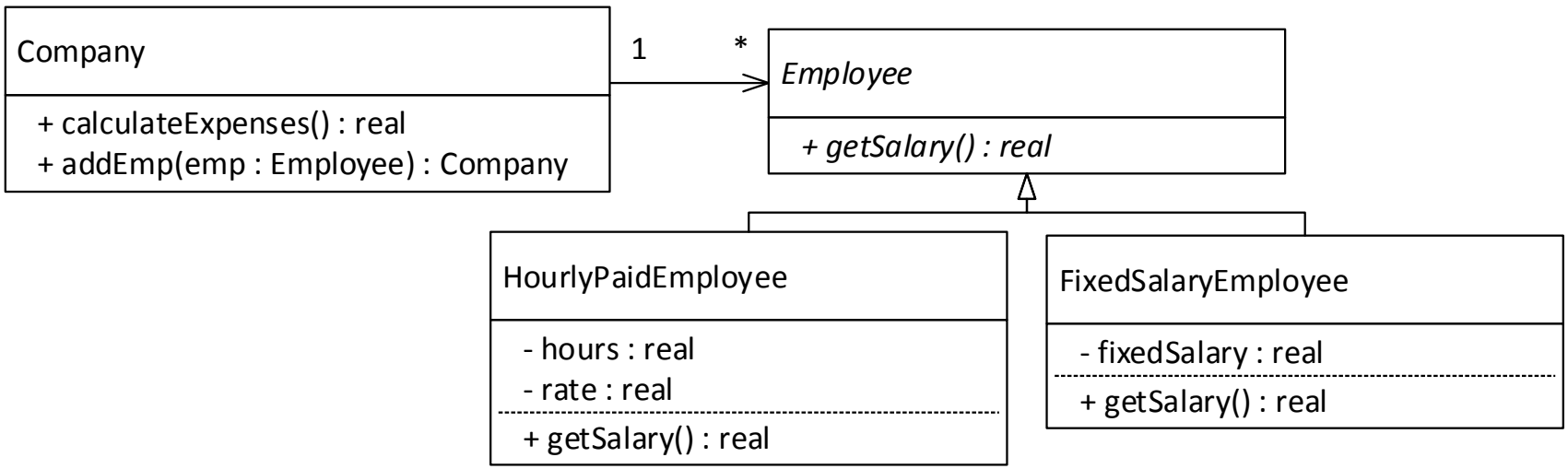
1. Backend: A configurable part producing the actual code
2. Library code: Utility code specific to a target language
3. Templates: Mappings of VDM nodes to a target language



The case study

UML description of the case study VDM model

A system for handling company salary expenses



The Company VDM class

VDM

```
class Company
instance variables
  private employees : seq of Employee;
operations
  ...
  public calculateExpenses: () ==> real
  calculateExpenses() == ...

  public addEmp : Employee ==> Company
  addEmp (emp) == ...
functions
  private start_calc: seq of Employee -> real
  start_calc(emps) == ...
end Company
```

The Company Java class

Java

```
...
public class Company {

 private List<Employee> employees;
 ...
 public double calculateExpenses() {...}

 public Company addEmp(Employee emp) {...}

 private double start_calc(List<Employee> emps) {...}

}
```

Adding employees to a company

VDM

```
public addEmp : Employee ==> Company
addEmp (emp) ==
(
 employees := employees ^ [emp];
 return self;
);
```

Java

```
public Company addEmp(Employee emp) {

  employees = Utils.seqConc(employees, Utils.seq(emp));
  return this;
}
```

Calculation of salary expenses (1/2)

VDM

```
private start_calc: seq of Employee -> real
start_calc(emps) ==
 if emps = []
  then 0
 else
  (hd emps).getSalary() + start_calc(tl emps);
```

Calculation of salary expenses (2/2)

Java

```
private double start_calc(List<Employee> emps) {

 if (emps.isEmpty())
 {
   return 0;
 }
 else
 {
    return emps.get(0).getSalary()
           + start_calc(emps.subList(1, emps.size()));
 }
}
```

Future plans

1. CG for a distributed hardware platform
2. Investigating extensibility of the CG
3. Mapping of union types
   - Difficult to generate but they easily appear in a model

```
--Type seq1 of (char) | nat1
if true then "one" else 2
--Type seq1 of (FixedSalaryEmployee | HourlyPaidEmployee)
[new FixedSalaryEmployee(), new HourlyPaidEmployee()]
```