An Architectural Evolution of the Overture Tool

Peter Jørgensen Kenneth Lausdahl
Peter Gorm Larsen

Aarhus University, Department of Engineering

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

Outline

History of the Overture tool (1/3)

1. Overture development started back in 2003
2. The initial work was primarily made by Master's students
3. The tool supported partial checking of static semantics
4. The syntax tree was stored in XML

History of the Overture tool (2/3)

Verhoef developed a tool for generation of the AST

1. The AST nodes were produced in Java and VDM
2. The Java nodes were used for developing the tool
3. The VDM nodes were used for developing tool extensions

History of the Overture tool (3/3)

1. At the same time VDMJ was being developed
2. Later VDMJ was integrated with Eclipse
3. The VDMJ integration resulted in two AST representations
4. It was possible to convert to the VDMJ AST
5. Why not make the generated AST compatible with VDMJ?

The VDMJ based AST architecture

1. Not intended for Overture integration
2. Handwritten AST nodes
3. Core functionality resides in the AST nodes
4. Tool extensions are likely to require AST modifications
5. Easier navigation in the AST would benefit IDE features

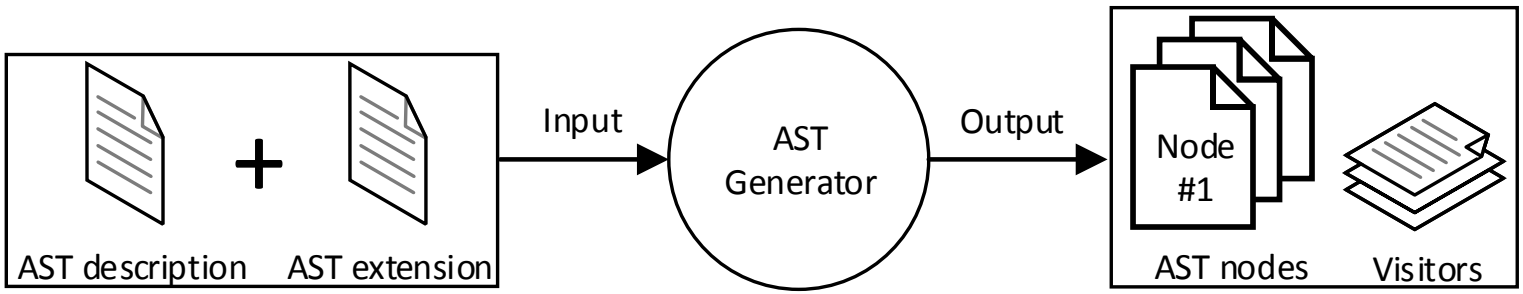
The new AST architecture in a nutshell

The new AST architecture introduces three major changes

- All non-trivial functionality is moved out of the nodes
- Nodes are being generated from a description
- The AST uses bidirectional node relations

```
// Find the type definition of "type"
type.getAncestor(ATypeDefinition.class)
```

How to extend the tree (1/2)



A new expression is added in the following way:

1. Update the AST description/AST extension
2. Update the parser for the new expression
3. Add the corresponding visitor cases

How to extend the tree (2/2)

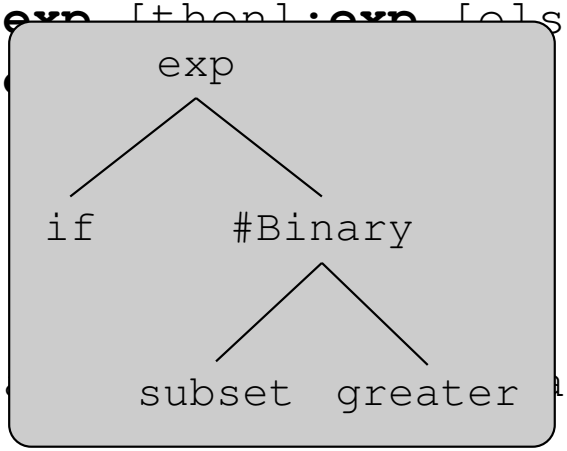
```
exp {-> package='org.overture.ast.expressions'}
 =   {if} [test]:exp [then]:exp [elseList]:elseIf*
          [else]:exp
 |   #Binary
 ...
 ;

#Binary {-> package='org.overture.ast.expressions'}
 =   {subset}
 |   {greater}
 ...
```

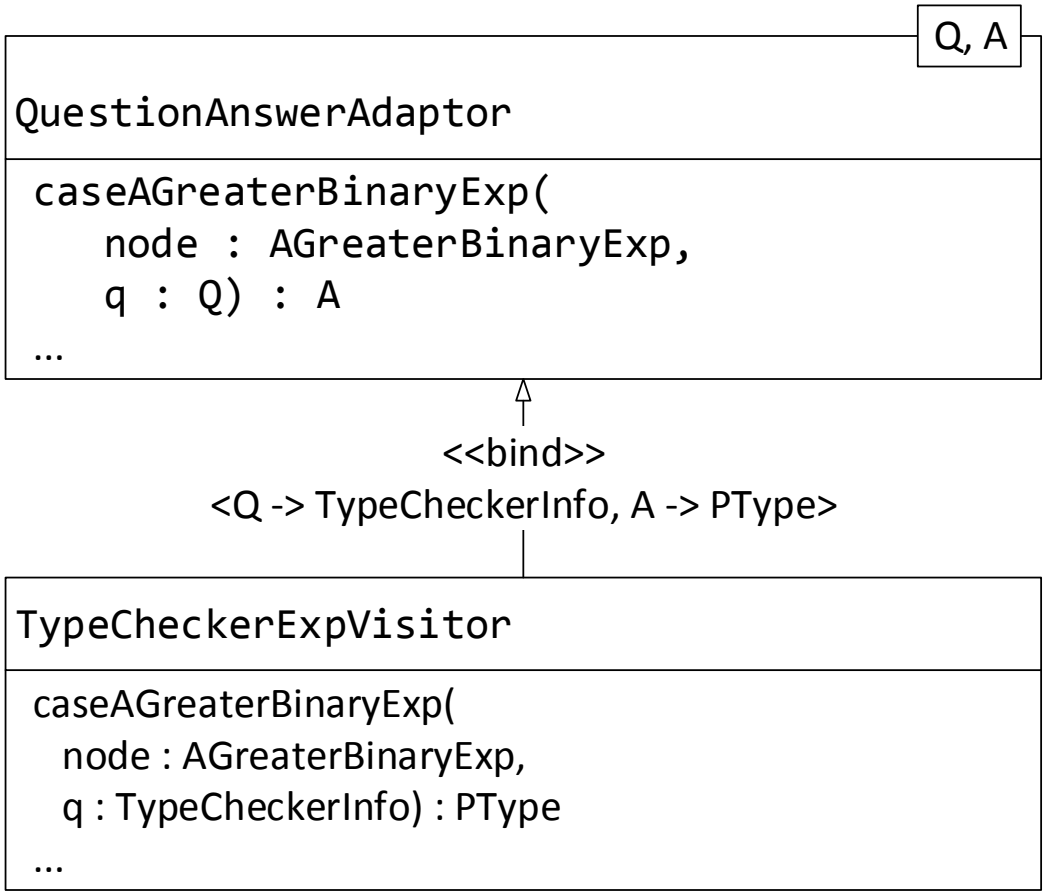
How to extend the tree (2/2)

```
exp {-> package='org.overture.ast.expressions'}
 =   {if} [test]:exp [then]:exp [elseList]:elseIf*
          [else]:
 |   #Binary
 ...
 ;

#Binary {-> pack                    ast.expressions'}
 =   {subset}
 |   {greater}
 ...
```



AST analysis is supported by visitor classes

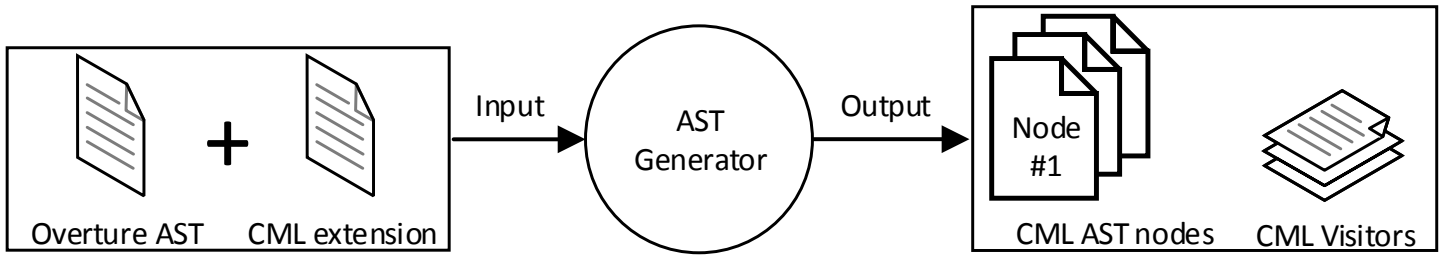


Type checking the “greater than” expression

```
public class TypeCheckerExpVisitor
extends QuestionAnswerAdaptor<TypeCheckInfo, PType> {
   //... Fields and visitor cases omitted
  @Override
  public PType caseAGreaterBinaryExp(
                AGreaterBinaryExp node,
                TypeCheckInfo q)
  {
    node.getLeft().apply(this, q);
    node.getRight().apply(this, q);
    ...
  }
}
```

Applications of the visitor based architecture

1. The COMPASS project
   - Continuous feedback for the new AST architecture



2. The new UML-VDM mapper
   - An AST is defined for the UML model
   - Mapping is done by converting between ASTs
   - The plugin makes heavy use of assistant classes

Future plans (1/2)

1. Enable development of new features
2. IDE modeling support
3. Generation of Overture components (e.g. parser)

Future plans (2/2)

Constructing a core interpreter

1. The current interpreter handles all three dialects of VDM
2. Have an abstract core interpreter
3. This future work item is speculative