Overture Architecture & Status

Peter Gorm Larsen & Marcel Verhoef

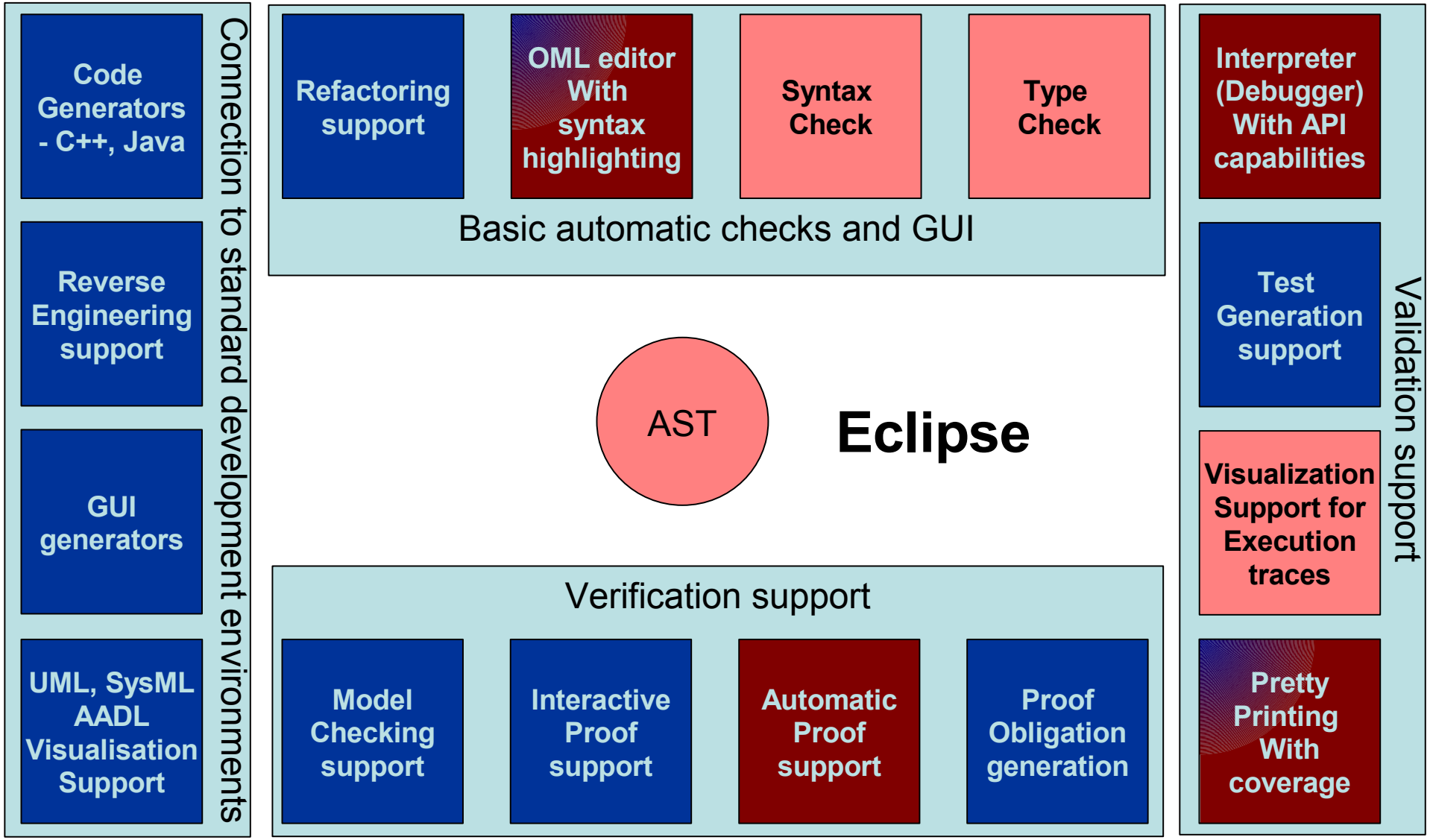
(updated 28 Nov 2006)

OVERTURE ARCHITECTURE

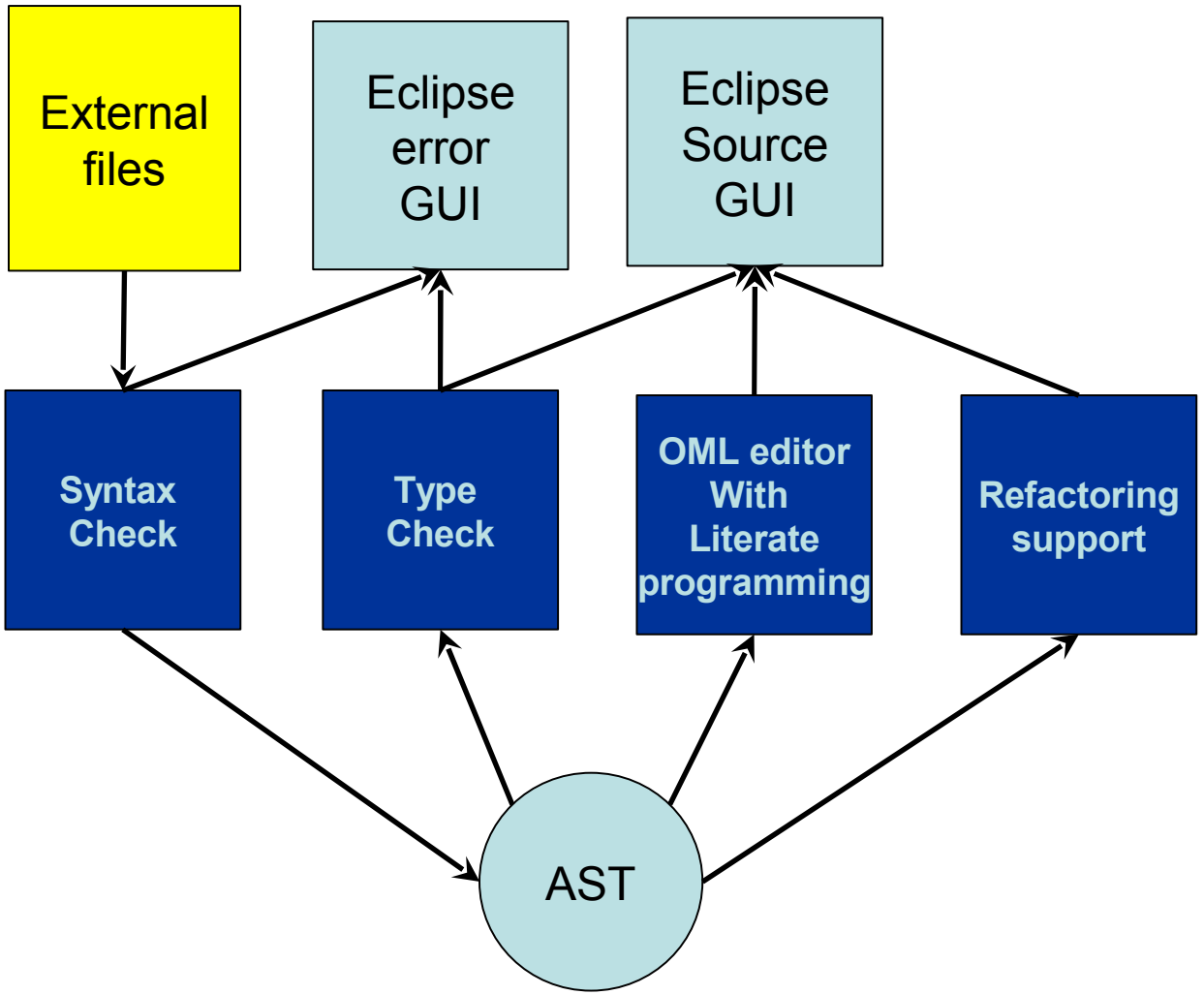
Overture versus VDMTools

- VDMTools (http://www.vdmtools.jp/en)
  - Closed source, proprietary (available under NDA)
  - Monolithic architecture (single binary), C++
  - Optimized for performance, industry strength
- Overture Tool project (http://www.overturetool.org)
  - Open source, GPL license
  - Plug-in architecture, Eclipse, Java
  - Optimized for flexibility, targets academic use
  - (partly) developed using VDMTools

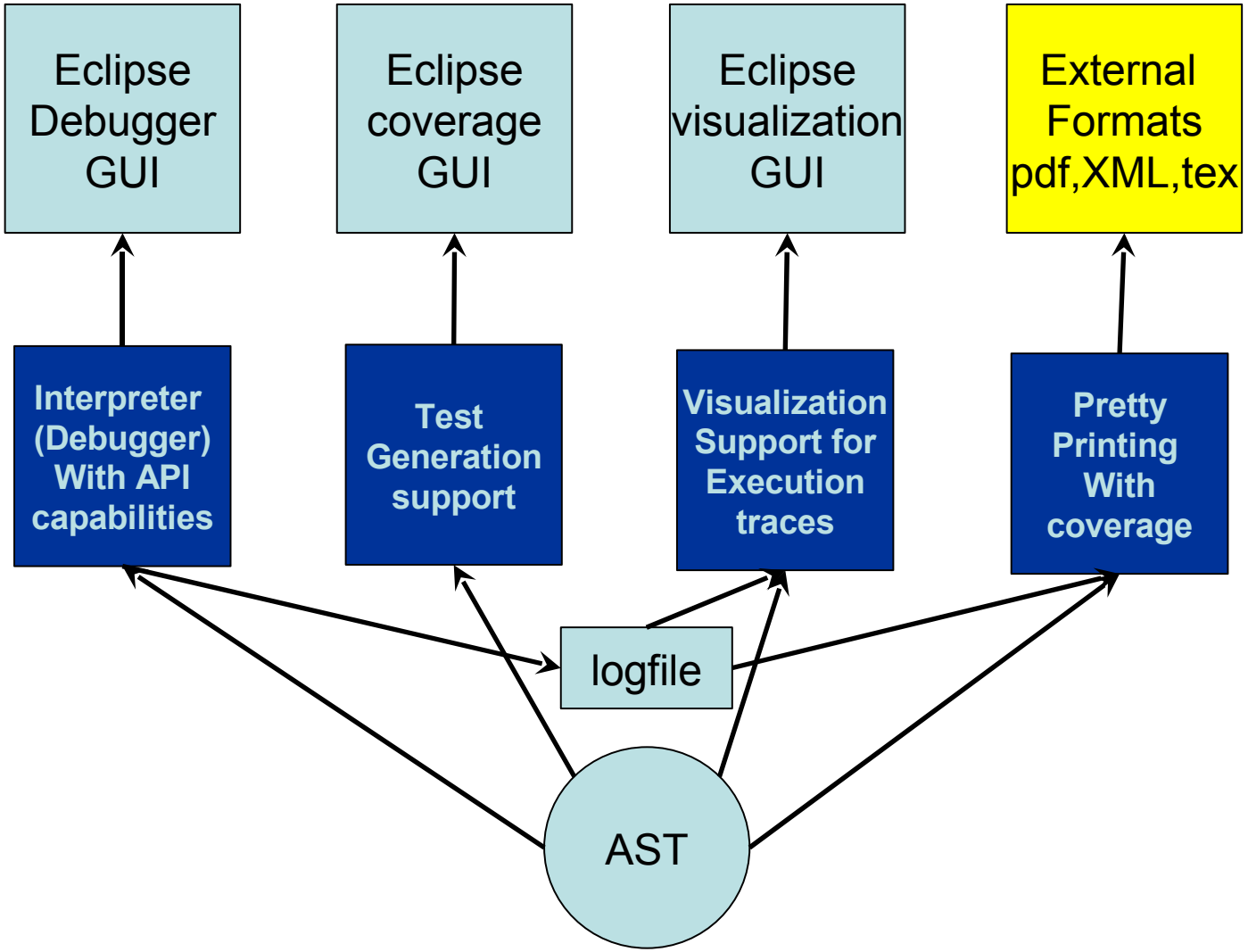
Overture Architecture Overview



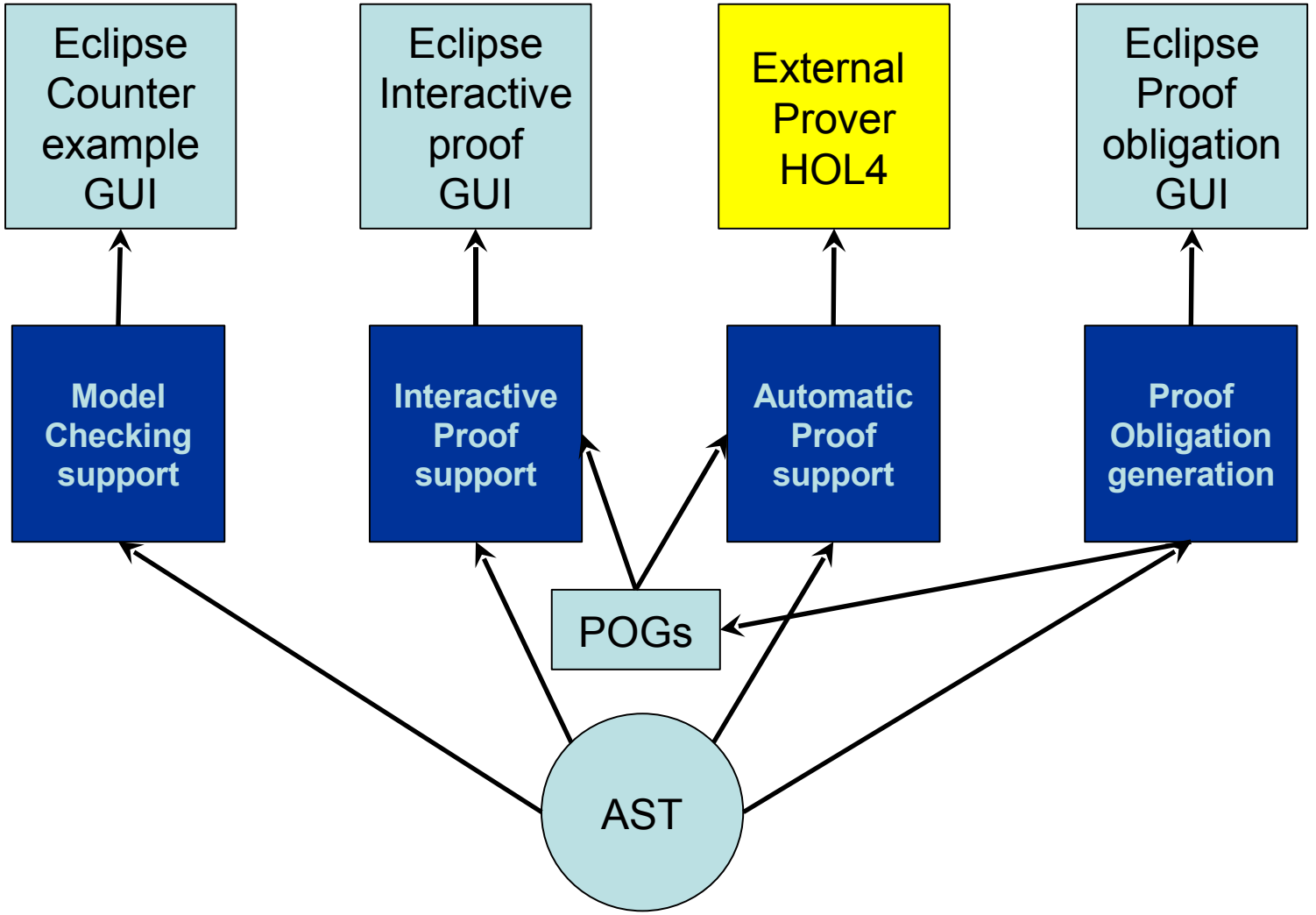
Basic automatic checks and GUI



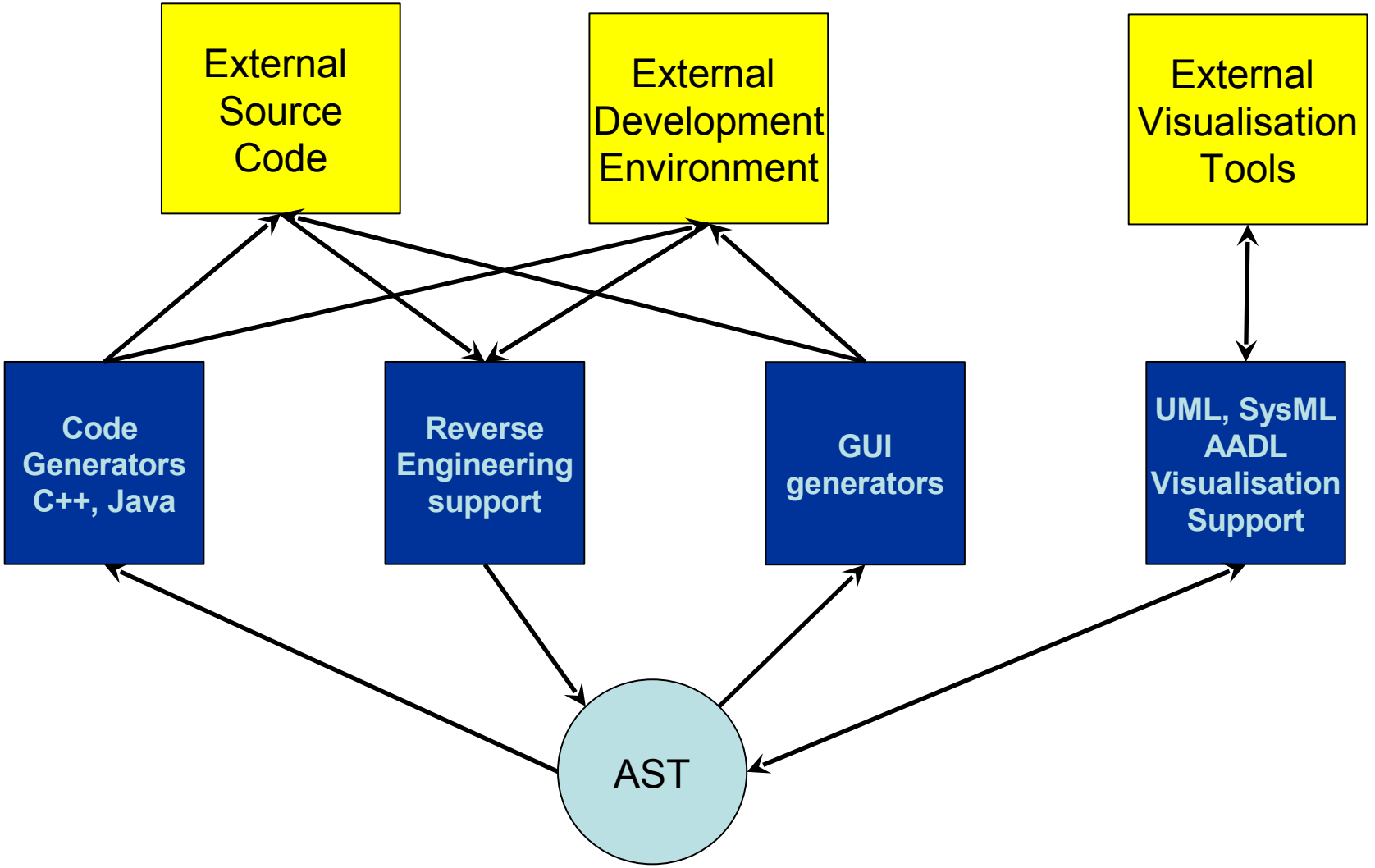
Validation support



Verification support



Other development environments



OVERTURE STATUS

Iteration One (2004)

- Pieter van der Spek (MSc, TU Delft)
  - Build parser and simple pretty printer
  - Experiments on improved error support in parser generator (published as ACM Sigplan Notices)
  - Delivered as Eclipse plug-in

  - *Limited (no) XML support*
  - *Direct manipulation of concrete syntax tree*

Iteration Two (2005)

- Jacob Porsborg Nielsen &
  Jens Kielsgaard Hansen (MSc, TU Denmark)
  - Re-implemented parser using ANTLR
  - abstract syntax with appropriate Java interfaces
  - XML support for reading / writing AST instances
  - Experimented with Eclipse architecture, many useful suggestions and prototype plug-ins

  - *Hand-coded AST implementation – very error prone*
  - *Many errors in parser implementation*

Meanwhile in 2006

- Address problem of AST maintenance
- Executive decision: we need a ***robust*** solution...
- ... to support language experiments
- Back To Basics: how can we re-use VDMTools parser code and know-how?
- How good are the open source ***jflex*** and ***byaccj*** tools?
- Early experiments performed
- Solution: automation is key
- “eat our own dog food”

Iteration Three (2006)

- Marcel Verhoef
  - Implemented AST generation tool + visitor support
  - Specified Overture AST in VDM-SL subset
  - Generate AST specification *and* implementation
  - Re-implemented parser using JFLEX & BYACCJ
  - Fixed problem in BJACCJ (Java 64k byte-code limit)
  - Implementation is 108% VDM++ compatible
  - (no local function definitions in let-expressions: -2 %)
  - (extensions to VICE are taken into account: +10 %)
  - Full UTF-8 file support (e.g. Japanese identifiers)

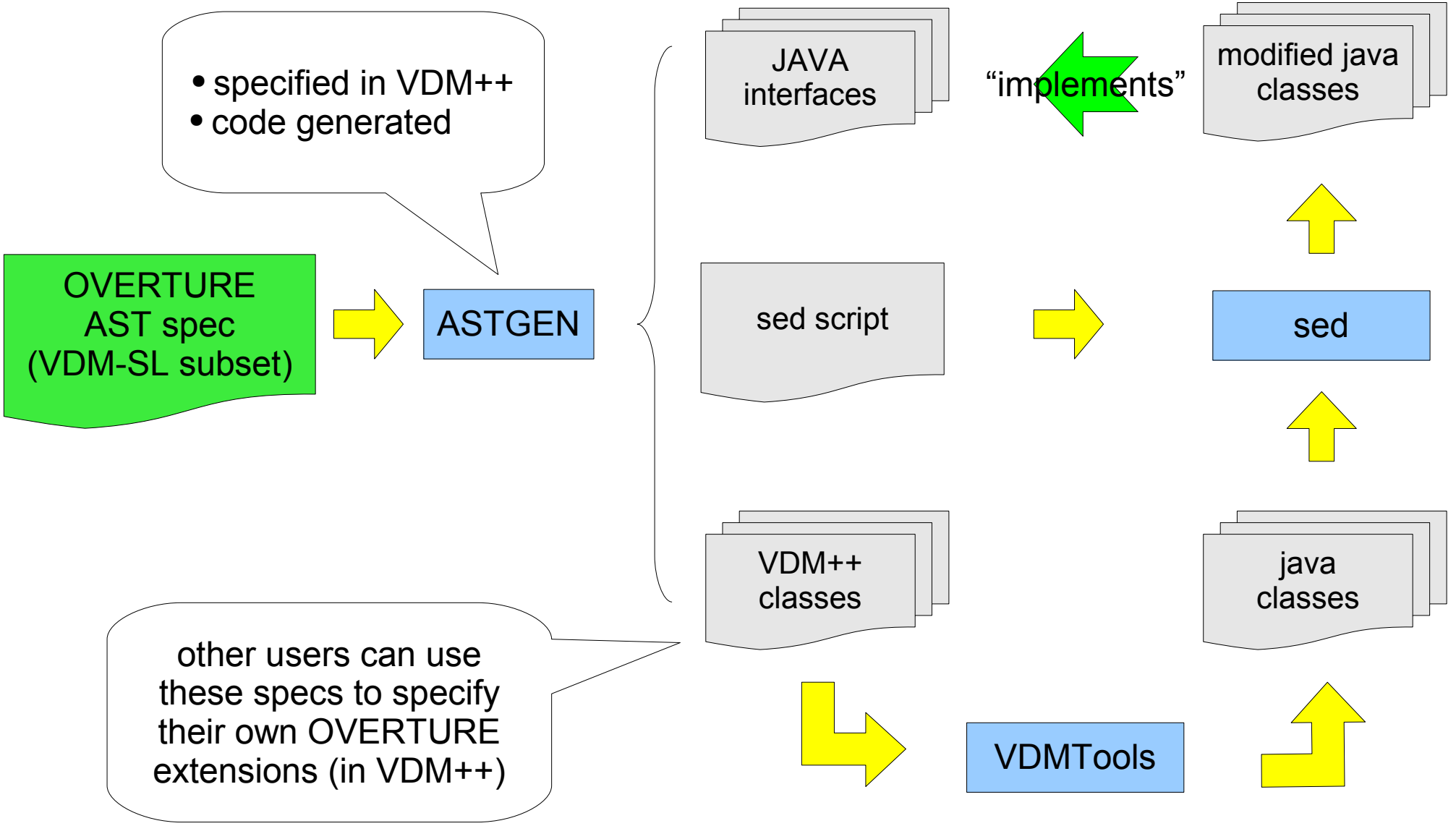
Iteration Three (2006)

- Checked using VDMTools test-suite (2000 test cases)
- Surprise: parser is quite fast (comparable to VDMTools)
- Default visitors are available for VDM-SL and VDM++
- Limited support for error recovery in BYACCJ
- To Do : improve build procedure (remove manual steps)

http://www.overturetool.org/twiki/bin/view/Main/OvertureParser

- Used by Thomas Christensen for static semantics
- Flexibility proven: short turn-around for new features

Automatic AST generation



AST specification (1)

```
Specifications ::
  class_list : seq of Class;

Class ::
  identifier          : Identifier
  inheritance_clause : [InheritanceClause]
  class_body          : seq of DefinitionBlock
  system_spec         : bool;

InheritanceClause ::
  identifier_list : seq of Identifier;

DefinitionBlock =
  TypeDefinitions |
  ValueDefinitions |
  FunctionDefinitions |
  OperationDefinitions |
  InstanceVariableDefinitions |
  SynchronizationDefinitions |
  ThreadDefinition;
```

AST specification (2)

```
Specifications ::
  class_list : seq of Class;

Class ::
  identifier          : Identifier
  generic_types       : seq of Type
  inheritance_clause  : [InheritanceClause]
  class_body          : seq of DefinitionBlock
  system_spec         : bool;

InheritanceClause ::
  identifier_list : seq of Identifier;

DefinitionBlock =
  TypeDefinitions |
  ValueDefinitions |
  FunctionDefinitions |
  OperationDefinitions |
  InstanceVariableDefinitions |
  SynchronizationDefinitions |
  ThreadDefinition;
```

AST specification (2)

```
Specifications ::
  class_list : seq of Class;

Class ::
  identifier          : Identifier
  generic_types       : seq of Type
  inheritance_clause  : [InheritanceClause]
  class_body          : seq of DefinitionBlock
  system_spec         : bool;

InheritanceClause ::
  identifier_list : seq of Identifier;

DefinitionBlock =
  TypeDefinitions |
  ValueDefinitions |
  FunctionDefinitions |
  OperationDefinitions |
  InstanceVariableDefinitions |
  SynchronizationDefinitions |
  ThreadDefinition;
```

Java interface (generated)

```
package org.overturetool.ast.itf;

import java.util.*;

public abstract interface IOmlClass extends IOmlNode
{
    abstract String getIdentifier();
    abstract Vector getGenericTypes();
    abstract IOmlInheritanceClause getInheritanceClause();
    abstract Boolean hasInheritanceClause();
    abstract Vector getClassBody();
    abstract Boolean getSystemSpec();
}
```

VDM++ AST “interface” classes (generated)

```
class IOmlClass is subclass of IOmlNode

operations
  public getIdentifier: () ==> seq of char
  getIdentifier() == is subclass responsibility;

  public getGenericTypes: () ==> seq of IOmlType
  getGenericTypes() == is subclass responsibility;

  public getInheritanceClause: () ==> IOmlInheritanceClause
  getInheritanceClause() == is subclass responsibility;

  public hasInheritanceClause: () ==> bool
  hasInheritanceClause () == is subclass responsibility;

  public getClassBody: () ==> seq of IOmlDefinitionBlock
  getClassBody() == is subclass responsibility;

  public getSystemSpec: () ==> bool
  getSystemSpec() == is subclass responsibility;

end IOmlClass
```

Abstract visitor support (generated)

```
class IOmlVisitor

operations
  public visitSpecifications: IOmlSpecifications ==> ()
  visitSpecifications (-) == is subclass responsibility;

  public visitClass: IOmlClass ==> ()
  visitClass (-) == is subclass responsibility;

  public visitInheritanceClause: IOmlInheritanceClause ==> ()
  visitInheritanceClause (-) == is subclass responsibility;

  public visitDefinitionBlock: IOmlDefinitionBlock ==> ()
  visitDefinitionBlock (-) == is subclass responsibility;

  ... operations for each AST element

end IOmlVisitor
```

Abstract visitor support (generated)

```
class IOmlVisitor

operations
  public visitSpecifications: IOmlSpecifications ==> ()
  visitSpecifications (-) == is subclass responsibility;

  public visitClass: IOmlClass ==> ()
  visitClass (-) == is subclass responsibility;

  public visitInheritanceClause: IOmlInheritanceClause ==> ()
  visitInheritanceClause (-) == is subclass responsibility;

  public visitDefinitionBlock: IOmlDefinitionBlock ==> ()
  visitDefinitionBlock (-) == is subclass responsibility;

  ... operations for each AST element

end IOmlVisitor
```

Concrete visitor support (generated)

```
public visitClass: IOmlClass ==> ()
visitClass(pcmp) ==
  ( dcl str : seq of char := prefix ^pcmp.identity() ^"(";
    pushNL();
    str := str ^getNL();
    printStringField(pcmp.getIdentifier());
    str := str ^result ^"," ^getNL();
    printSeqofField(pcmp.getGenericTypes());
    str := str ^result ^"," ^getNL();
    if pcmp.hasInheritanceClause()
    then printNodeField(pcmp.getInheritanceClause())
    else result := "nil";
    str := str ^result ^"," ^getNL();
    printSeqofField(pcmp.getClassBody());
    str := str ^result ^"," ^getNL();
    printBoolField(pcmp.getSystemSpec());
    str := str ^result;
    popNL();
    str := str ^getNL() ^")";
    result := str );
```

VDM++ AST “implementation” (generated)

```
class OmlClass is subclass of IOmlClass

operations
  public identity: () ==> seq of char
  identity () == return "Class";

  public accept: IOmlVisitor ==> ()
  accept (pVisitor) == pVisitor.visitClass(self);

  public OmlClass:
      (seq of char) *
      (seq of IOmlType) *
      [IOmlInheritanceClause] *
      (seq of IOmlDefinitionBlock) *
      (bool) ==> OmlClass
  OmlClass (p1,p2,p3,p4,p5) ==
   ( setIdentifier(p1);
     setGenericTypes(p2);
     setInheritanceClause(p3);
     setClassBody(p4);
     setSystemSpec(p5) );
```

VDM++ AST “implementation” (generated)

```
...

instance variables
  private ivInheritanceClause : [IOmlInheritanceClause] := nil

operations
  public getInheritanceClause: () ==> IOmlInheritanceClause
  getInheritanceClause() ==
    return ivInheritanceClause
    pre hasInheritanceClause();

  public hasInheritanceClause: () ==> bool
  hasInheritanceClause () ==
    return ivInheritanceClause <> nil;

  public setInheritanceClause: [ IOmlInheritanceClause ] ==> ()
  setInheritanceClause(parg) ==
    ivInheritanceClause := parg;

...
```

(b)yacc(j) grammar

```
Class:
    ClassHeader Identifier GenericTypeList InheritanceClause
    DefinitionBlock LEX_END Identifier

{ OmlClass res;
  OvertureLexem header = (OvertureLexem) $1;
  OvertureLexem nm1 = (OvertureLexem) $2;
  Vector theGenerics = (Vector) $3;
  OmlInheritanceClause theClause = (OmlInheritanceClause) $4;
  Vector theClassBody = (Vector) $5;
  OvertureLexem nm2 = (OvertureLexem) $7;
  if ( checkClassName(nm1,nm2) ) {
    res = new OmlClass();
    res.setSystemSpec(checkClassHeader(header));
    res.setIdentifier(lexemToString(nm1));
    res.setGenericTypes(theGenerics);
    res.setInheritanceClause(theClause);
    res.setClassBody(theClassBody);
  } else {
    res = null;
  }
  $$ = res; }
```

Overture parser at work

```
let a in set {1,...,10} in a + 1
```

```
mave@PC514 /cygdrive/d/projects/Overture Parser
$ ./ovparse -sl -O expr1.vpp
Start test parser
Reading file "expr1.vpp"
Encoding = Cp1252
0 error(s) found
End test parser

mave@PC514 /cygdrive/d/projects/Overture Parser
$ ./ovparse -pp -O expr1.vpp
Start test parser
Reading file "expr1.vpp"
Encoding = Cp1252
0 error(s) found
End test parser
```

Overture parser – VDM-SL output

```
mk_ LetBeExpression(
  mk_ SetBind(
    [
      mk_ PatternIdentifier("a")
    ],
    mk_ SetRangeExpression(
      mk_ SymbolicLiteralExpression(mk_ NumericLiteral(1)),
      mk_ SymbolicLiteralExpression(mk_ NumericLiteral(10))
    )
  ),
  nil,
  mk_ BinaryExpression(
    mk_ Name(
      nil,
      "a"
    ),
    <PLUS>,
    mk_ SymbolicLiteralExpression(mk_ NumericLiteral(1))
  )
)
```

Overture parser – VDM++ output

```
new OmlLetBeExpression(
  new OmlSetBind(
    [
      new OmlPatternIdentifier("a")
    ],
    new OmlSetRangeExpression(
      new OmlSymbolicLiteralExpression(new OmlNumericLiteral(1)),
      new OmlSymbolicLiteralExpression(new OmlNumericLiteral(10))
    )
  ),
  nil,
  new OmlBinaryExpression(
    new OmlName(
      nil,
      "a"
    ),
    new OmlBinaryOperator(28),
    new OmlSymbolicLiteralExpression(new OmlNumericLiteral(1))
  )
)
```

Sexy Enough?

- Approach followed is NOT restricted to a specific parser generator
- Requirement: any Overture parser shall conform to the AST interface definition
- Current implementation is the base line

Support for language experiments

- Generic recipe to follow:
  - Change the AST definition
  - Re-generate the AST (AstGen & VDMTools)
  - Modify the scanner / parser (jflex, byaccj)
  - Recompile java code
- Turn-around time:
  - 2 hours (minor changes)
  - 1 day (larger changes)

Support for tool development (1)

- The (preferred) VDM++ recipe
  - Take the AST VDM++ “interfaces” (IOml*) as is
  - Take VDM++ class “OmlVisitor”
  - Refactor (rename) this class
  - Specify the required functionality directly in VDM++
  - Validate the specification using VDMTools
  - Generate the Java implementation using VDMTools
  - Compile and integrate into Eclipse plug-in

Support for tool development (2)

- Alternate Java recipe
  - Take the AST Java interface classes
  - Take the OmlVisitor.java code template
  - Refactor (rename) this class
  - Write your tool directly in Java
  - Compile and integrate into Eclipse plug-in

Next Steps & Future Work

- Complete position information functionality (required by some downstream tools)
- Build parser plug-in for Eclipse and on-the-fly syntax high-lighting in Eclipse editor
- Build simple pretty-print plug-in (for LaTeX) and “refactor” the AST generation tool
- Explicit AST version management and migration (through OMG MOF and using Eclipse EMF?)
- Create XML schema and read / write operations (visitor)