An Integrated Verification Environment for JML: Architecture and Early Results

Patrice Chalin, Perry R. James, and George Karabotsos

Dependable Software Research Group
Computer Science and Software Engineering Dept.
Concordia University, Montreal, Canada
{chalin, perry, george}@dsrg.org

Road map

- JML: language & tools
- Requirements for Next-generation
- JML4 features
- JML4 architecture
- Early benefits

Java Modelling Language

- A language for describing behavior of code
- Tools to ensure they match

Java Modelling Language

- A language for describing behavior of code
  - DbC with lightweight specs
  - Full BISL with heavyweight specs
- Tools to ensure they match

Java Modelling Language

- A language for describing behavior of code
- Tools to ensure they match

– RAC	JML compiler (`jmlc`)
– ESC	ESC/Java2
– FSPV	LOOP, JACK
– testing	JmlUnit
– doc	JmlDoc
– autogen	JmlSpec, Daikon, Houdini

Current State of affairs: Limitations of current tools

Lots of good tools... but

- Not interoperable
- Own parsers, desugarers, etc.
- Out of date
  - Java 5 released in September 2004
  - Still no support for generics
- Mostly command-line driven

Current State of affairs: What worked well in JML 2

- Common JML tool suite
  - Checker, RAC, JmlUnit
- Built on MultiJava compiler (MJ)
  - MJ mostly independent of JML
  - JML subclasses MJ classes & overrides methods
  - Extension points
    - Calls to empty methods

This idea used in JML 4

Requirements for any Next-generation JML tools

- Remove duplication of effort
  - Tool developers
  - Analysis
- No maintenance of a Java compiler
- Integrated (development and) Verification Environment (IVE)
  - Support RAC, ESC, and FSPV

JML4 achieves these

JML4

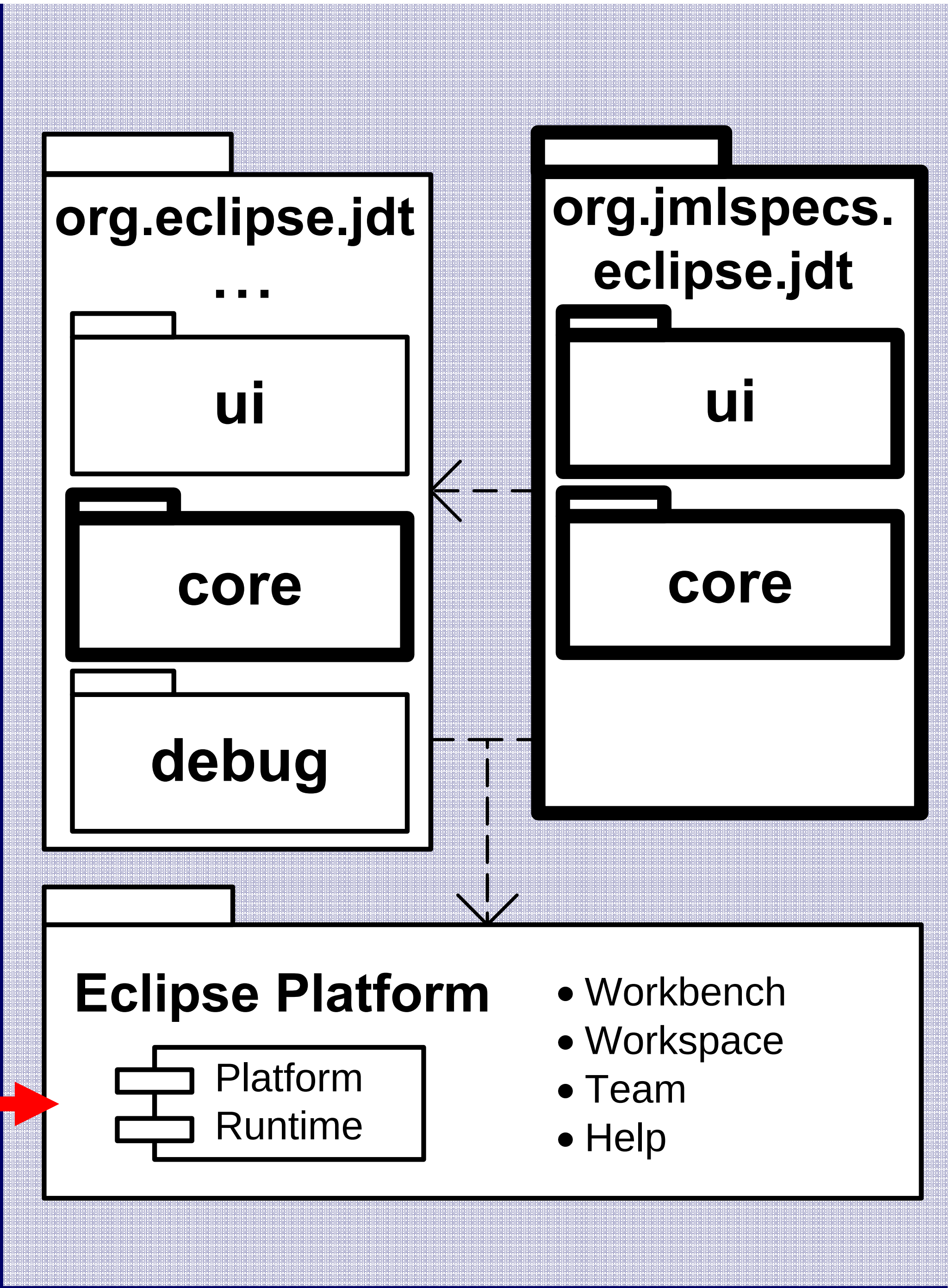
- Built atop Eclipse, integrated with the JDT
- Currently supports
  - Processes annotations in .java and .jml files
  - Non-null type system
    - Static enforcement
    - RAC generation (desugared)
  - Initial Design by Contract
    - Initial integration with ESC/Java 2
    - RAC generation

High-level Package view

JML 4 replacement for JDT plug-in
+ additional UI plug-in

JML 4 changes / introduces packages in **bold**

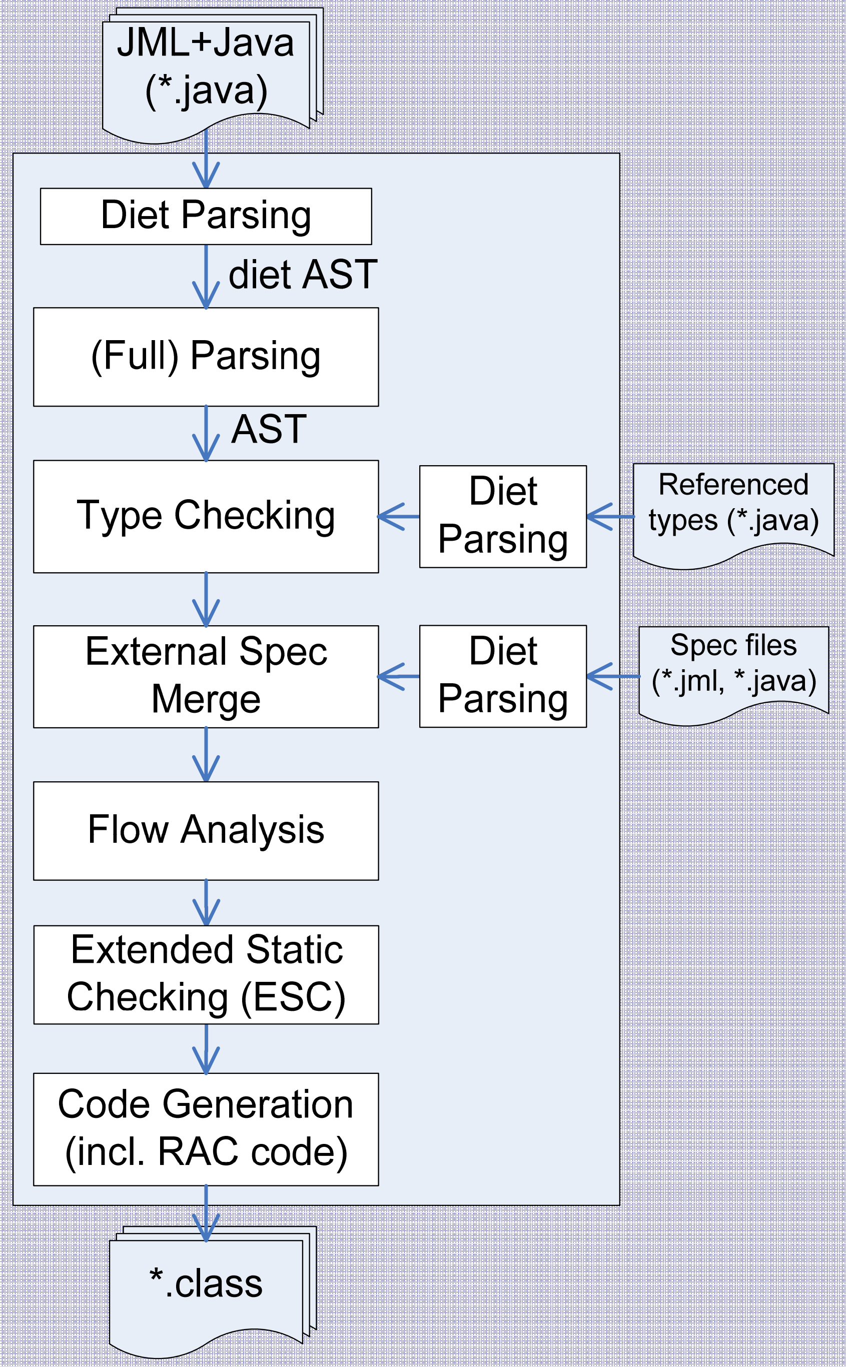
Everything's a plug-in ! (except this small bit)



Compilation Phases

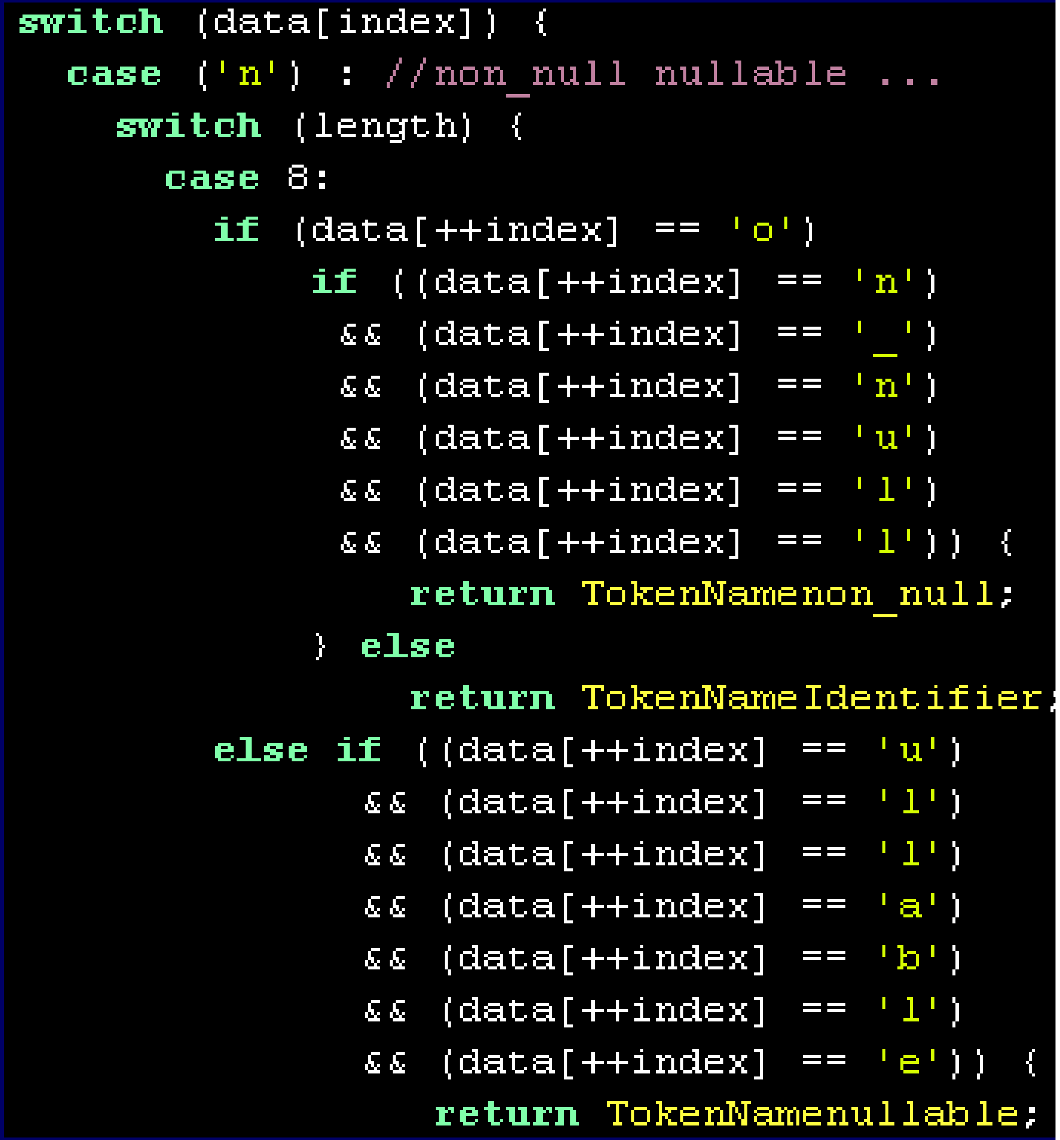
Inline & external specs processed

Static verification before code generation so it can influence runtime checking



Eclipse JDT: Lexical analysis

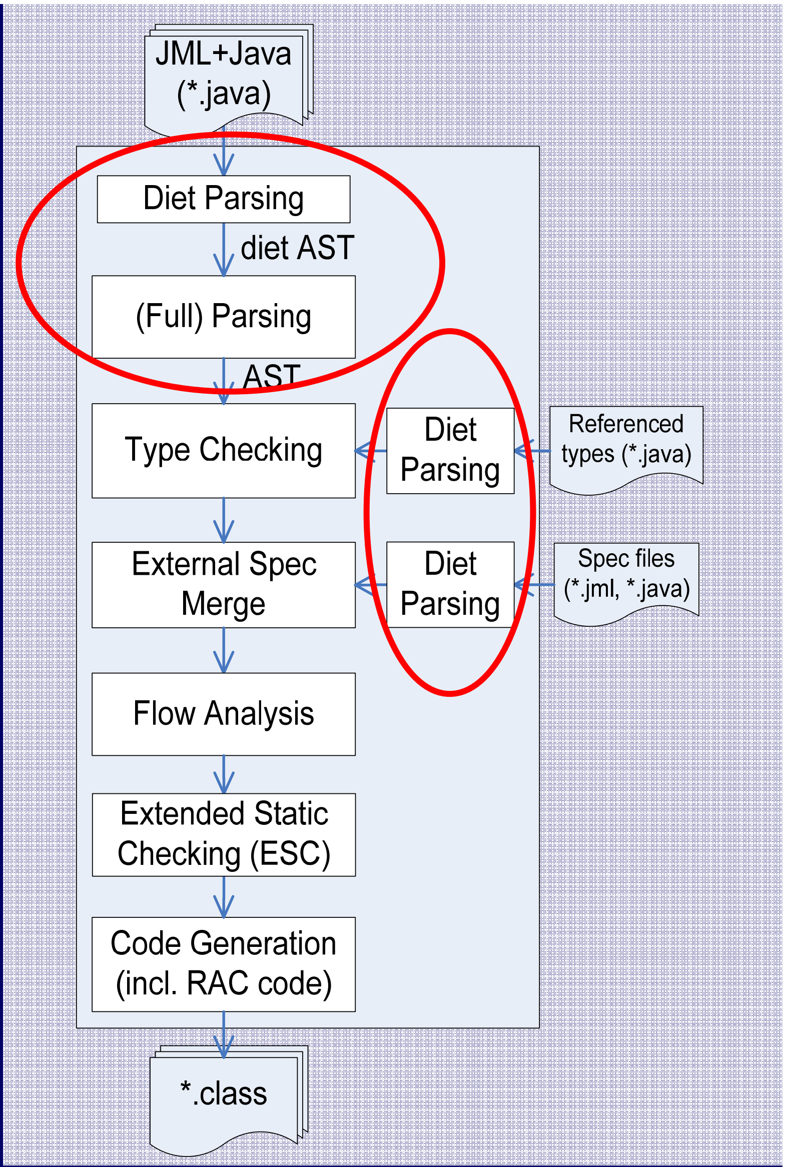
- Hand crafted
- Tedious to modify keywords
- JML in special comments
  - Easy to switch to augmented keywords



```
switch (data[index]) {
  case ('n') : //non_null nullable ...
    switch (length) {
      case 8:
        if (data[++index] == 'o')
            if ((data[++index] == 'n')
             && (data[++index] == '_')
             && (data[++index] == 'n')
             && (data[++index] == 'u')
             && (data[++index] == 'l')
             && (data[++index] == 'l')) {
                return TokenNamenon_null;
            } else
                return TokenNameIdentifier;
        else if ((data[++index] == 'u')
              && (data[++index] == 'l')
              && (data[++index] == 'l')
              && (data[++index] == 'a')
              && (data[++index] == 'b')
              && (data[++index] == 'l')
              && (data[++index] == 'e')) {
                 return TokenNamenullable;
```

Eclipse JDT: Parsing – 2 kinds

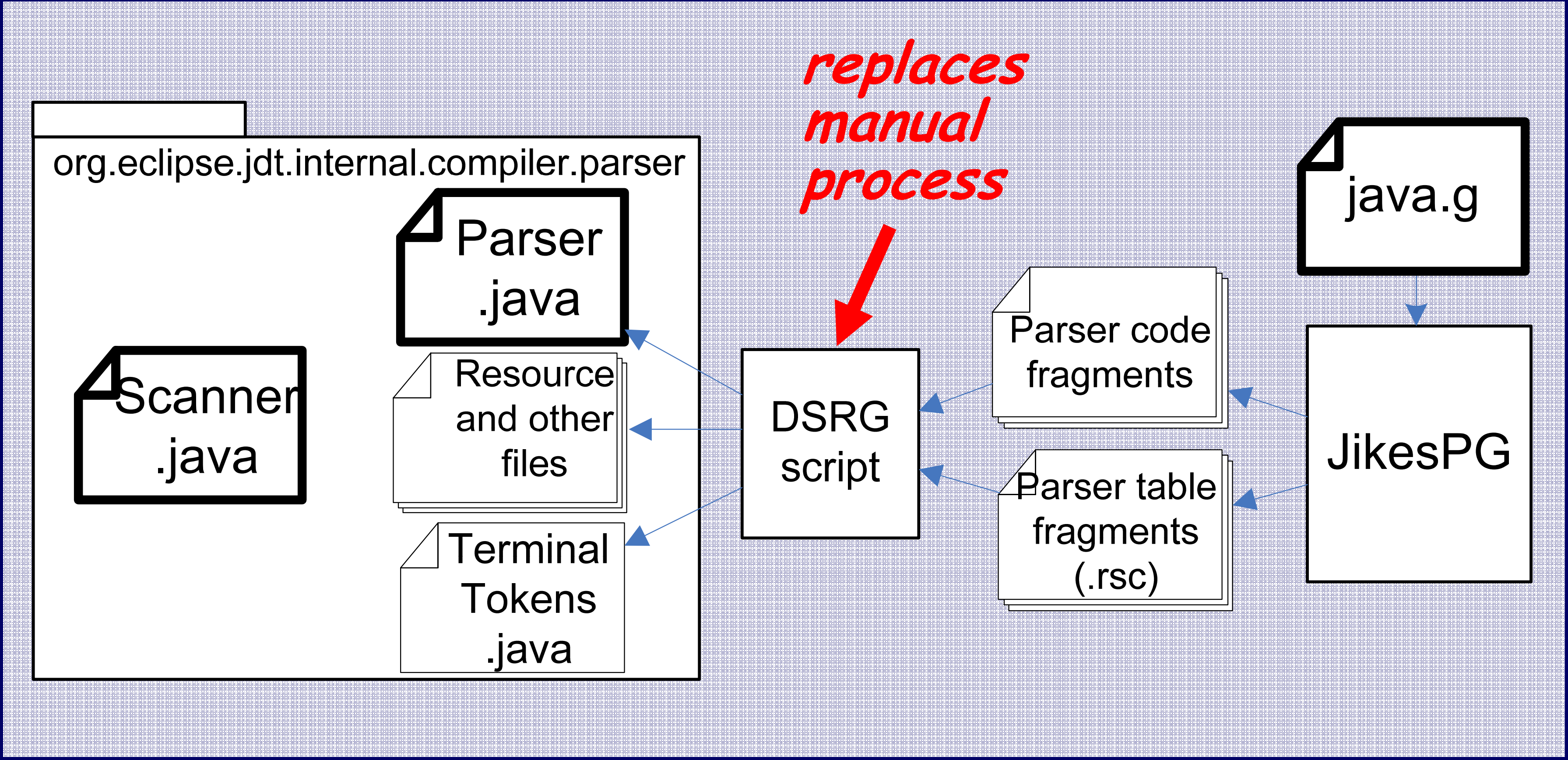
- Diet parsing
  - Method bodies skipped
  - Only signature information
- Full parsing
  - Method bodies processed
  - All info available
- For memory efficiency
  - All diet parsed
  - Full parsed individually, then discarded



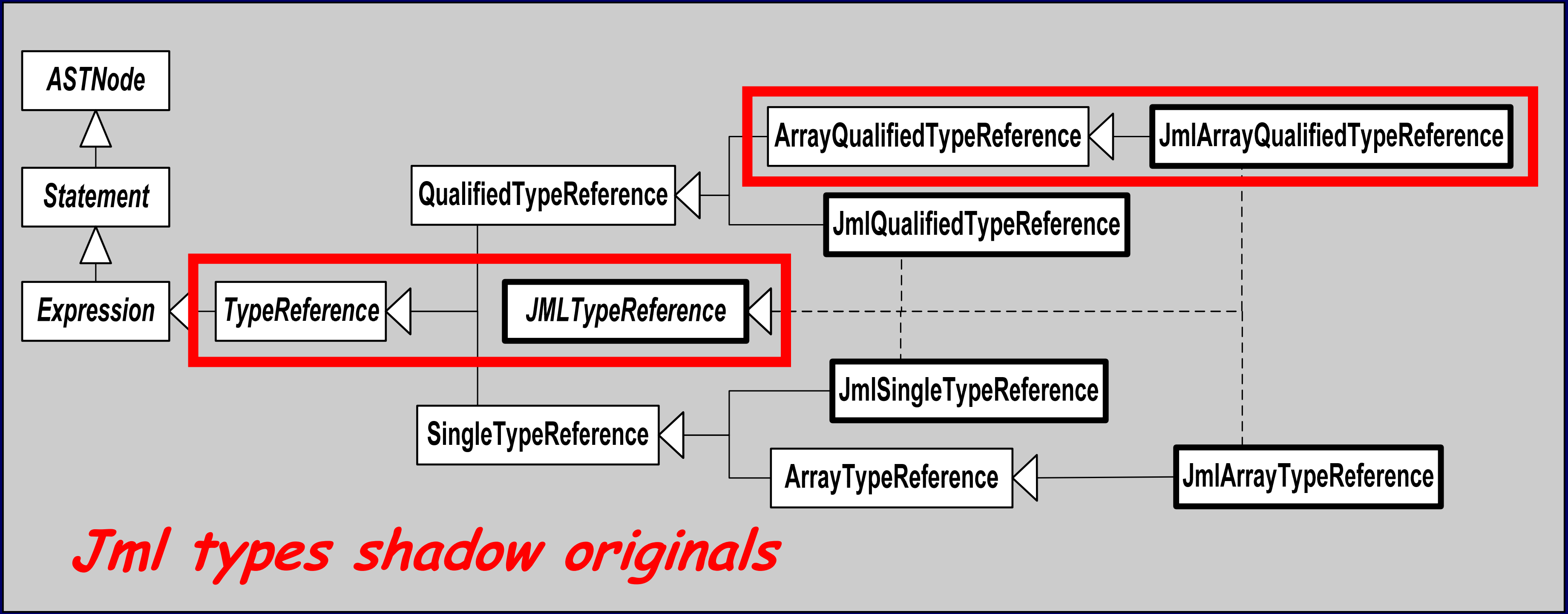
Eclipse JDT: Parsing

+ Parser generated using JikesPG
+ Grammar follows Java Language Specification
+ One semantic action per reduction
- Little support for token stacks
- Replaced calls to ASTNode constructors with JML-specific versions
- Documentation only in German

Eclipse JDT: Customizing the lexer and parser



Eclipse JDT: Part of the AST hierarchy



- No copy & change of code
- Only overriding & hooks

Eclipse JDT: Type checking & Flow analysis

- Changed to support non-null type system
- Extended with hooks (calls to empty methods) added in original `resolve` and `analyseCode` methods

Eclipse JDT: Static verification

- Originally delegated to ESC/Java2
- Now working to use
  - Eclipse as a front end
  - ESC/Java2 back end
- Later steps are to
  - Optionally remove RAC for proved properties
  - Add interface for FSPV

Eclipse JDT: RAC code generation (part of a hook)

```
public static void generateNullityTest(
                          CodeStream codeStream,
                          String exceptionType,
                          String msg) {
    BranchLabel nonnullLabel =
            new BranchLabel(codeStream);
    codeStream.dup();
    codeStream.ifnonnull(nonnullLabel);
    codeStream.newClassFromName(exceptionType,
    codeStream.athrow();
    nonnullLabel.place();
}
```

JML 4 Validation

- Compiler is kept up to date with new features
  - JDT already supports Java 6
- No copy & change of JDT code
  - use subclassing and method extension points
  - bracketing our changes with special comments
- CVS vendor branches
- Merging in weekly updates is painless
  - takes on average < 10 min.

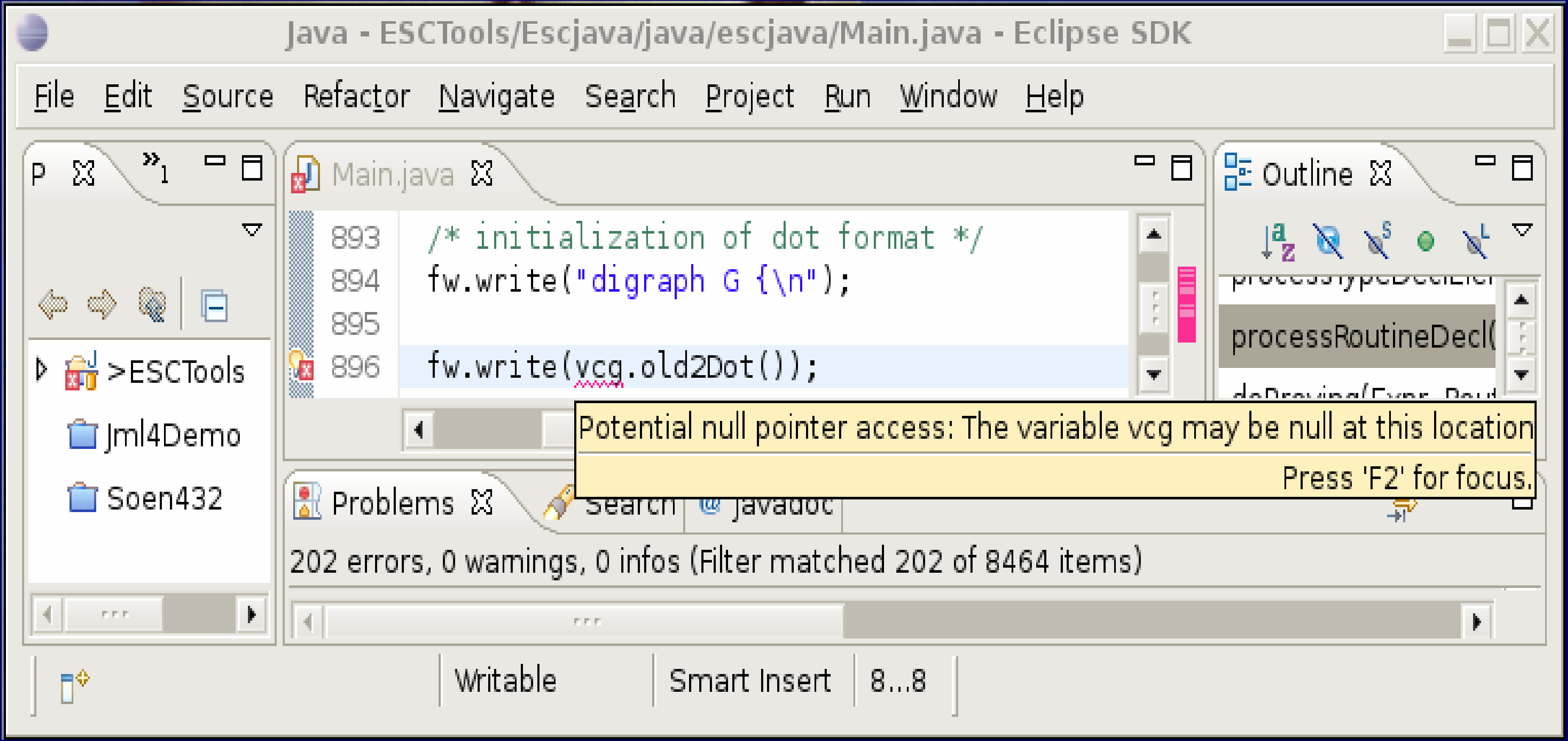
JML 4: Early benefits

- Ran JML 4 on ESC/Java2
- New problems found in Main class

JML 4: Early benefits

```
VcGenerator vcg = null; ...
try {
  ... // possible assignment to vcg
}
// multiple catch blocks
catch (Exception e) {
  ...
}
...
fw.write(vcg.old2Dot()); // possible NPE
```

JML 4: Early benefits


Java - ESCTools/Escjava/java/escjava/Main.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Main.java
Outline
processRoutineDecl(
>ESCTools
Jml4Demo
Soen432
Problems
Search
Javadoc
Writable
Smart Insert
8...8

```
/* initialization of dot format */
fw.write("digraph G {\n");

fw.write(vcg.old2Dot());
```

Potential null pointer access: The variable vcg may be null at this location

Press 'F2' for focus.

202 errors, 0 warnings, 0 infos (Filter matched 202 of 8464 items)

JML 4: Early benefits

fe.Options

In a superclass of Main

```
  static public Options options = null;
```

In Main

```
 public static Options options() {
       return (Options)options;
 }
```

esc.Options

250+ occurrences of

```
      options().someField
```

or

```
      options().someMethod()
```

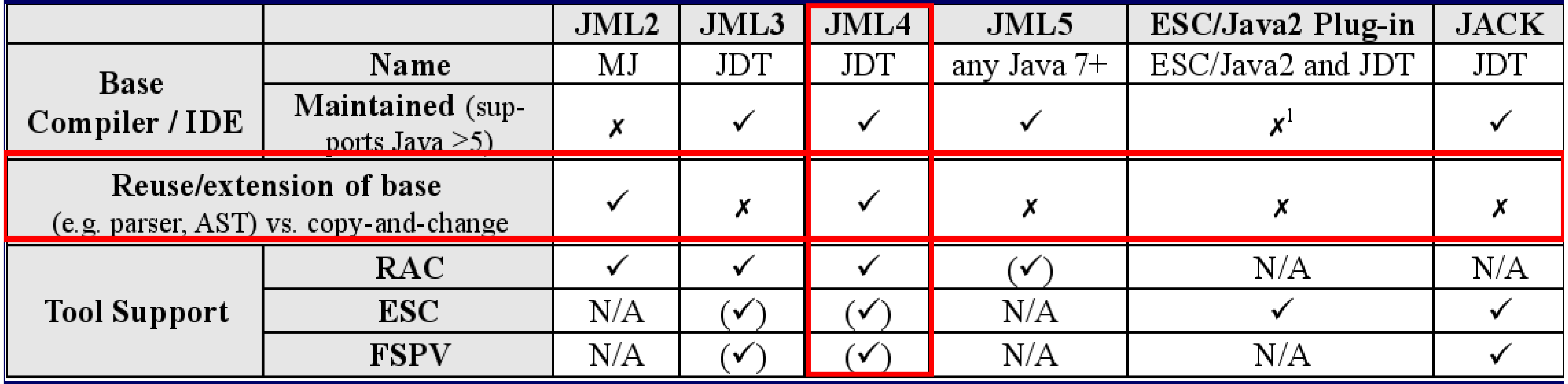
JML4: Next steps

- Continue adding support JML level 0 (and above)
- Enhance ESC support
- Include interface for FSPV
- …

Related work

- JML 3
  - A proper plug-in → doesn’t use non-API classes
  - **Needs its own parser, type checker, etc.**
- JML 5
  - Specifications in ‘@’ annotations
  - Can’t put annotations everywhere we want
  - **Needs its own parser, type checker, etc.**

Related work



		JML2	JML3	JML4	JML5	ESC/Java2 Plug-in	JACK
Base Compiler / IDE	**Name**	MJ	JDT	JDT	any Java 7+	ESC/Java2 and JDT	JDT
	Maintained (supports Java >5)	✗	✓	✓	✓	✗[1]	✓
Reuse/extension of base (e.g. parser, AST) vs. copy-and-change		✓	✗	✓	✗	✗	✗
Tool Support	**RAC**	✓	✓	✓	(✓)	N/A	N/A
	ESC	N/A	(✓)	(✓)	N/A	✓	✓
	FSPV	N/A	(✓)	(✓)	N/A	N/A	✓

Conclusion

- Integrated (development and) Verification Environment (IVE)
- Support RAC, ESC, and FSPV
- No need to maintain a Java compiler
- Unify support to remove duplication of effort

An Integrated Verification Environment for JML

Thank you !