Building an IDE with Rascal

Mark Hills

CWI & INRIA ATEAMS

18 May 2011
Setting the Stage
Outline

1. Setting the Stage
2. Parsing
Outline

1. Setting the Stage
2. Parsing
3. Outliners and Annotators
4. Contributors
Outline

1. Setting the Stage
2. Parsing
3. Outliners and Annotators
4. Contributors
5. Conclusions
Building on Past Work

- GIPE and GIPE II: Centaur (LeLisp, Prolog)
- ASF+SDF (Lisp, then C, with Java front-end)
- Rascal (C and Java, now completely in Java), building on the Eclipse IDE Meta-Tooling Platform (Eclipse IMP) for language IDE support
Running Example: Oberon-0

- A subset of Oberon, a successor to Pascal and Modula-2
- Developed as part of a language workbench competition
- Includes common, basic features from many languages: variables, constants, procedures, arrays, records, simple control flow constructs
- Goal was to develop a number of language tools: editor, type checker, compiler, etc
A Swap Procedure in Oberon-0

PROCEDURE Swap(VAR x, y: INTEGER);
VAR
    temp: INTEGER;
BEGIN
    temp := x;
    x := y;
    y := temp
END Swap;
Arrays and Procedures in Oberon-0

MODULE testL4;
VAR
  x: ARRAY 4 OF INTEGER;
  i: INTEGER;

PROCEDURE f(i: INTEGER; z: ARRAY 4 OF INTEGER);
BEGIN
  Write(z[i]); WriteLn()
END f;

BEGIN
  i := 0;
  WHILE i < 4 DO
    x[i] := i; f(i,x);
    i := i + 1
  END
END testL4.
Grammars defined using Rascal grammar definition notation

A Rascal program then builds a Java-based parser for the grammar

Parser is GLL – filtering rules used to remove ambiguities
Example: Oberon-0 Grammar

```plaintext
syntax Statement
  = assign: Ident var "::=" Expression exp
  |
  ifThen: "IF" Expression condition "THEN"
    {Statement ";"}+ body
    ElsIfPart*
    ElsPart?
    "END"
  |
  whileDo: "WHILE" Expression condition "DO"
    {Statement ";"}+ body
    "END"

;
Rascal Meta-Programming Architecture

Grammar Source Code
Parser Generator
AST Generator
AST Source Code
Parser Source Code
Java Compiler
AST Builder Source Code
Interpreter Source Code
Rascal Programs
Rascal Parser
Parse Tree
AST Builder
AST
Rascal Interpreter
Input Output

Legend:
build-time flow
run-time flow
data
operation

Build-time
Run-time

Legend:

Hills
Building an IDE with Rascal
9/23
Outliners in IDEs

- Outlines provide a quick overview of code, indicating which constructs (classes, methods, functions, variables, etc) have been defined
- Outlines also provide a way to browse the code quickly – selecting an element in the outline takes the programmer to the appropriate part of the code
Code Outlining Example: Java in Eclipse

```java
private final IValueFactory values;

public IO(IValueFactory values){
    super();
    this.values = values;
}

public void print(IValue org, IEvaluatorContext eval){
    PrintWriter currentOutStream = eval.getOutputStream();
    synchronized(currentOutStream){
        try{
            if(org.getType().isStringType()){
                currentOutStream.println(((IString)org).getValue().toString());
            } else if(org.getType().isSubtypeOf(Factory.Tree)){
                currentOutStream.println(TreeAdapter.yield((IConstructor)org));
            } else{
                currentOutStream.println(org.toString());
            }
        } finally{
            currentOutStream.flush();
        }
    }
}

public void iPrint(IValue org, IEvaluatorContext eval){
    PrintWriter currentOutStream = eval.getOutputStream();
    StandardTextWriter w = new StandardTextWriter(true, 2);
    synchronized(currentOutStream){
        try{
            print(w, org, eval);
        } finally{
            w.flush();
        }
    }
}
```
Outlining Support in Rascal: Building the Outline

- Outlines are built over the concrete syntax of a language
- Labels indicate the display name in the outline view
- Locations allow the user to jump to the outlined item

```java
public node outlineModule(Module x) {
    return outlineDecls(x.decls)[@label="<x.name>"];
}
```

```java
Node outlineDecls(Declarations decls) {
    cds = outline([ constDecl()[@label="<cd.name>"][@loc=cd@loc] |
    /ConstDecl cd := decls.consts ])[@label="Constants"];
    tds = outline([ typeDecl()[@label="<td.name>"][@loc=td@loc] |
    /TypeDecl td := decls.types ])[@label="Types"];
    vds = outline([ varDecl()[@label="<vd.names>"][@loc=vd@loc] |
    /VarDecl vd := decls.vars ])[@label="Variables"];
    return outline([cds, tds, vds]);
}
```
registerOutliner registers an outliner function with the IDE

The IDE then calls this function to build the outline automatically as the file changes

The IDE also provides the outline view, using the location and name info to build the view content

registerOutliner("l4", outlineModule);
Code Outlining Example: Oberon-0 in Rascal

```
PROCEDURE Multiply;
  VAR x, y, z: INTEGER;
BEGIN
  Read(x);
  Read(y);
  z := 0;
  WHILE x > 0 DO
    IF x MOD 2 = 1 THEN
      z := z + y
    END;
    y := 2*y;
    (* Dag *)
    x := x DIV 2 END;
  Write(x);
  Write(y);
  Write(z);
  WriteLn
END Multiply;

PROCEDURE Divide;
  VAR x, (*. Q *) y, r, q, w: INTEGER;
BEGIN
```

Hills Building an IDE with Rascal

14/23
Annotators allow annotations to be added to language constructs and displayed in the editor

Typical examples: name resolution, type checking – want errors to be displayed graphically to users, marking error locations

```rascal
public Module checkModule(Module x) {
    m = implode(x);
    <m, st> = resolve(m);
    errors = { error(l, s) | <l, s> <- st.scopeErrors };  
    if (errors == {}) {
        errors = check(m, st.symbolTable);
    }
    return x[@messages = errors];
}

registerAnnotator("l4", checkModule);
```
Annotator Example: Type Checking Oberon-0

```plaintext
MODULE Collatz;

VAR even, odd : INTEGER;

PROCEDURE doCollatz();
VAR current : INTEGER;
currentEven : BOOLEAN;

PROCEDURE computeEven();
BEGIN
  IF current MOD 2 = 0 THEN
    currentEven := even
  ELSE
    currentEven := odd
  END
END computeEven;
```
Contributors provide a way to add more advanced functionality

Each contribution is a menu item – execution is triggered by the user

Examples: interaction with external tools, compilation, visualization
An Example Contributors Menu

```
BEGIN
  even := 1;
  odd := 0;
  doCollatz();
END Collatz.

BEGIN
  printSequence();
END printSequence;
```

- Run As
- Debug As
- Validate
- Team
- Compare With
- Replace With
- Pretty Print
- WikiText
- Preferences...
- Oberon
  - Compile to C
  - Compile to Java
  - Format
  - Obfuscate (protect your precious oberon0 code!)
  - Show control flow graphs
  - Compile to Java bytecode and run
Visualization Contribution: Control Flow Graph
Contributors: Integration with External Tools

- Contributors in Rascal-based IDEs are not limited to those written in Rascal
- Example: linking a Rascal-based front-end with a Maude-based analysis framework

![Diagram showing integration process between Rascal and K/Maude](image-url)
Contributors: Integration with External Tools

Information from the external tool can be used to set up annotations...
... and to add other information, such as entries in an Eclipse Problems view.
Building on IMP, Rascal provides a number of hooks to add support for language IDEs.

Support based on higher-level constructs in Rascal: instead of generating from a language specification, Rascal provides abstractions for working with programming languages and programs, providing high degree of customizability.

Bridge to Java allows IDE features to be based on tools written in Rascal and/or Java and on external tools.