EASY Programming with Rascal

Paul Klint

Universiteit van Amsterdam

CWI

Institut National de Recherche en Informatique et en Automatique INRIA

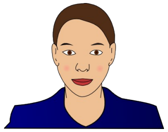
centre de recherche LILLE - NORD EUROPE

Joint work with
Emilie Balland, Bas Basten, Jeroen van den Bos, Mark Hills, Arnold Lankamp, Bert Lisser, Tijs van der Storm, Jurgen Vinju

Opening BLDL, November 4, 2009, Bergen, Norway

Cast of Heroes

- Alice, system administrator
- Bernd, forensic investigator
- Charlotte, financial engineer
- Daniel, multi-core specialist
- Elisabeth, model-driven engineering specialist



Meet Alice

- Alice is security administrator at a large online marketplace
- Objective: look for security breaches
- Solution:
  - Extract relevant information from system log files, e.g. failed login attempts in Secure Shell
  - Extract IP address, login name, frequency, …
  - Synthesize a security report

Meet Bernd

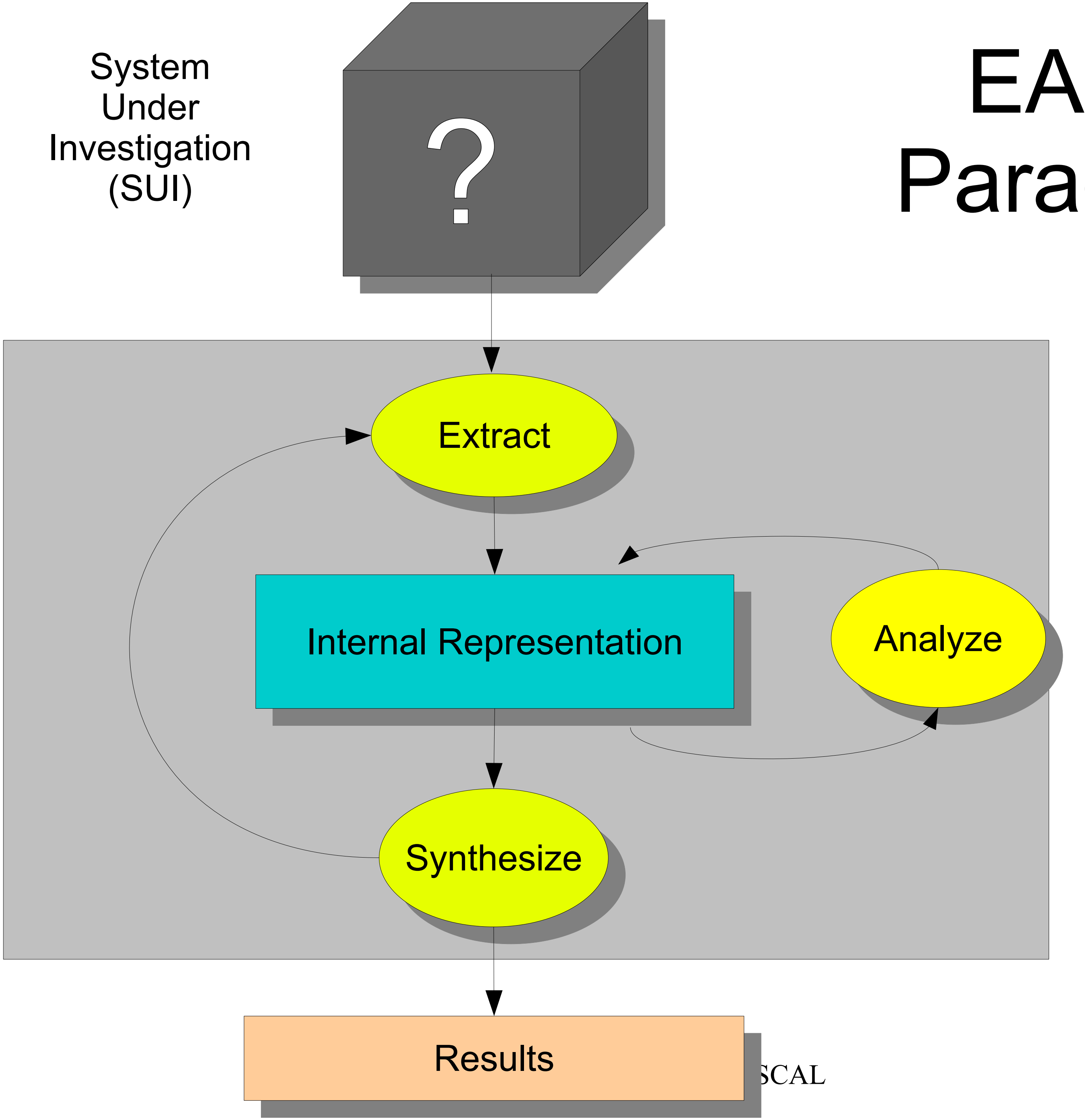
- Bernd: investigator at German forensic lab
- Objective: finding common patterns in confiscated digital information in many different formats. This is very labor intensive.
- Solution:
    - design DERRICK a domain-specific language for this type of investigation
    - Extract data, analyze the used data formats and synthesize Java code to do the actual investigation

What are their Common Problems?

- How to parse source code/data files
- How to extract facts from it
- How to perform computations on these facts
- How to generate new source code
- How to synthesize other information

EASY: Extract-Analyze-SYnthesize Paradigm

EASY Paradigm



What tools are available to our heroes?

- Lexical tools: Grep, Awk, Perl, Python, Ruby
  - Regular expressions have limited expressivity
  - Hard to maintain
- Compiler tools: yacc, bison, CUP, ANTLR
  - Only automate front-end part
  - Everything else programmed in C, Java, ..
- Attribute Grammar tools: FNC2, JastAdd, …
  - Mostly analysis, weak in transformation

What tools are available to our heroes?

- Relational Analysis tools: Grok, Rscript
  - Strong in analysis
- Transformation tools: ASF+SDF, Stratego, TOM, TXL
  - Strong in transformation
- Logic languages: Prolog
- Many others …

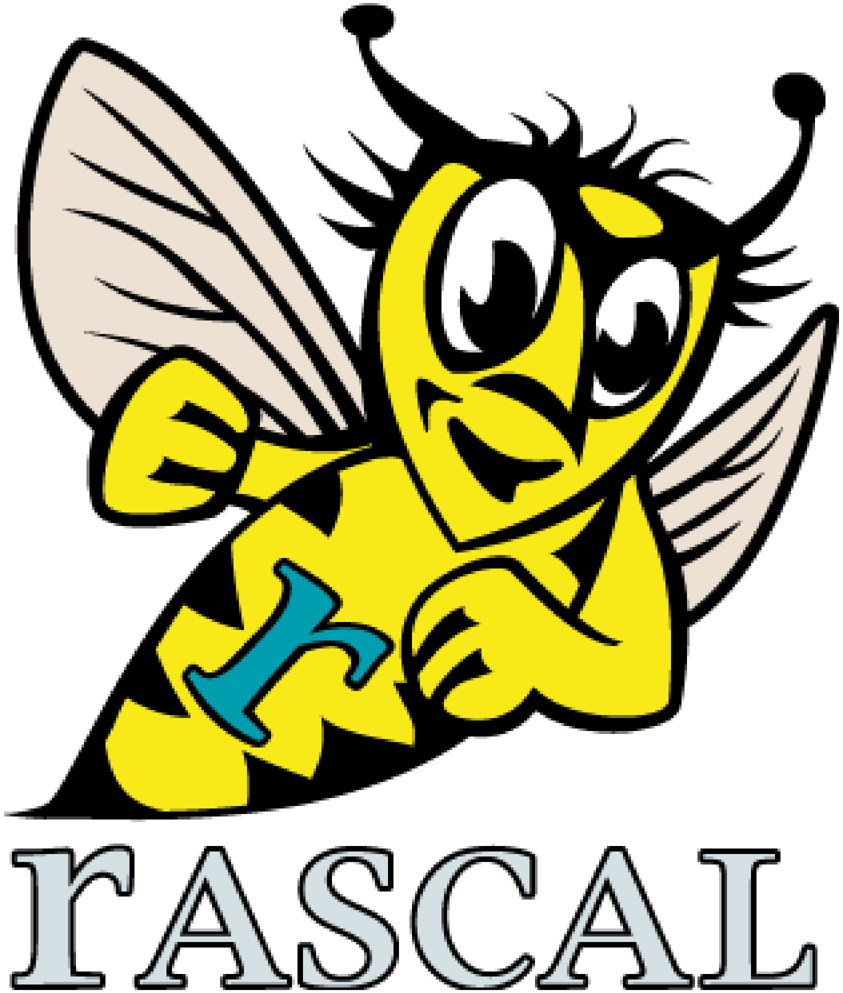
Apologies if your favorite tool does not appear in this list

	Extract	Analyze	Synthesize
Lexical tools	++	+/-	--
Compiler tools	++	+/-	+/-
Attribute grammar tools	++	+/-	--
Relational tools	--	++	--
Transformation tools	--	+/-	++
Rascal	++	++	++

Why a new Language?

- No current technology spans the full range of EASY steps
- There are many fine technologies but they are
  - highly specialized
  - hard to learn
  - not integrated with a standard IDE
  - hard to extend
  - ...

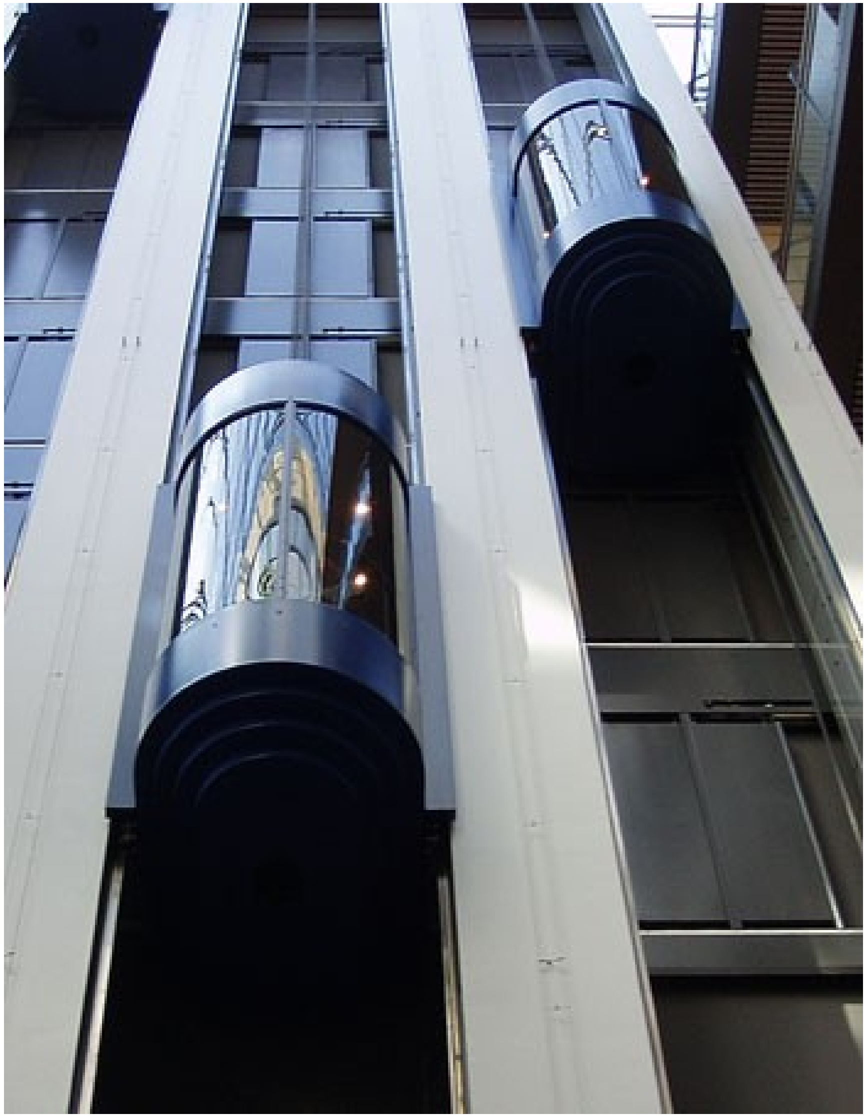
Here comes Rascal to the Rescue



Rascal ...

- ... is a new language for meta-programming
- … supports the EASY paradigm
- ... is based on
  - Syntax Analysis
  - Term Rewriting
  - Relational Calculus

Rascal Elevator Pitch



Rascal Elevator Pitch

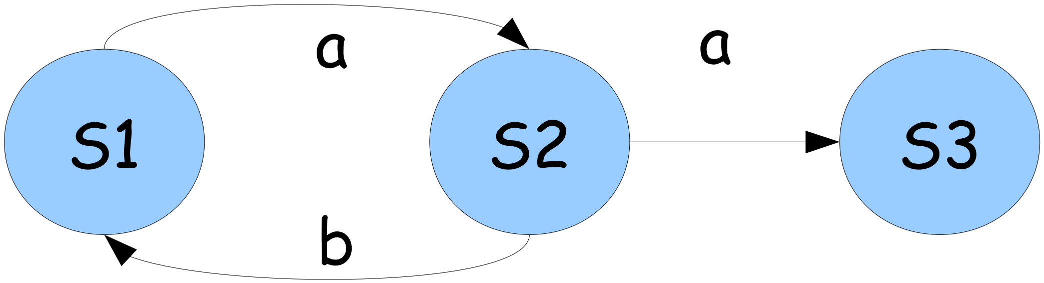
- Sophisticated built-in data types
- Immutable data
- Static safety
- Generic types
- Local type inference
- Pattern Matching
- Syntax definitions and parsing
- Concrete syntax
- Visiting/traversal
- Comprehensions
- Higher-order
- Familiar syntax
- Java and Eclipse integration
- Read-Eval-Print (REPL)

HOW EXCITING!! PLEASE TELL ME MORE!!

Example

A Domain-specific Language for State Machines

State Machine



```
canReach(
    finite-state machine
    state S1;
    state S2;
    state S3;
    trans a: S1 -> S2;
    trans b: S2 -> S1;
    trans a: S2 -> S3
)   = ( S1 : {S1, S2, S3},
        S2: {S1, S2, S3},
        S3: {})
```

State Machine Concrete Syntax

```
module demo/StateMachine/Syntax
...
    "state" Id                                      -> State
    "trans" Id ":" Id  "->" Id                      -> Trans
    State                                           -> Decl
    Trans                                           -> Decl
   “finite-state” “machine”  {Decl ";"}+            -> FSM
```

CanReach (1)

```
module demo::StateMachine::CanReach

import demo::StateMachine::Syntax;
import Graph;

FSM example =
        finite-state machine
          state S1;
          state S2;
          state S3;
          trans a: S1 -> S2;
          trans b: S2 -> S1;
          trans a: S2 -> S3;
… (next sheet)
```

A **concrete**, **unquoted**, FSM text fragment.

CanReach (2)

```
module demo::StateMachine::CanReach
… (previous sheet)

public map[str, set[str]] canReach(FSM fsm){
   transitions = getTransitions(fsm);
   closure = transitions+;
   return ( s : closure[s] |  str s <- carrier(transitions) );
}
```

Extract transitions as a graph

Transitive closure

Map comprehension

Enumerate all states

return a map in which each state is associated with all states that can be reached from it

CanReach (3)

```
module demo::StateMachine::CanReach
… (previous sheet)

public graph[str] getTransitions(FSM fsm){
  return {
    { < “<from>”, “<to>” > |
                /` trans <Id a>: <Id from> -> <Id to>` <- fsm
    }
}
```

Enumerate all transitions in the FSM

Convert a tree element to a string

Concrete pattern with variables

Example

Generating Getters and Setters

Generating Getters and Setters

- Given:
  - A class name
  - A mapping from names to types

Required:

  - Generate the named class with getters and setters

Generating getters and setters: Input

```
public map[str, str] fields = (
   "name" : "String",
   "age" : "Integer",
   "address" : "String"
);
```

Field name of type String

Field age of type Integer

Field address of type String

```
  genClass("Person", fields)
```

Generate class person with these fields

Generting getters and setters Expect Output

```
public class Person {
        private Integer age;
        public void setAge(Integer age) {  this.age = age;   }
        public Integer getAge() { return age;   }

        private String name;
        public void setName(String name) { this.name = name;   }
        public String getName() {  return name;   }

        private String address;
        public void setAddress(String address) {
                                                    this.address = address;   }
        public String getAddress() {  return address;   }
}
```

Generating Getters and Setters

```
public str genClass(str name, map[str,str] fields) {
 return "
   public class <name > {
     <for (f <- fields) {
       str t = fields[f];
       str n = capitalize(f);>
       private <t> <f>;
       public void set<n>(<t> <f>) { this.<f> = <f>;  }
       public <t> get<n>() { return <f>;  }
     <}>
   }
";
}
```

String with computed interpolations

Red is interpolated

Blue is literal

Example

Fact extraction and visualization

While working on a Java project ...

- For example, jspwiki
- What are the different file types used in this project?

What are the file types in this project?

```
module demo::filetypes
import Resources;
import viz::Chart;

public void main(){
    jspwiki = getProject(|project://jspwiki|);
    extensions = ();
    visit(jspwiki){
        case file(loc l): extensions[l.extension]? 0 += 1;
    }
    pieChart("Extensions", extensions, dim3());
}
```

Access to Eclipse resources

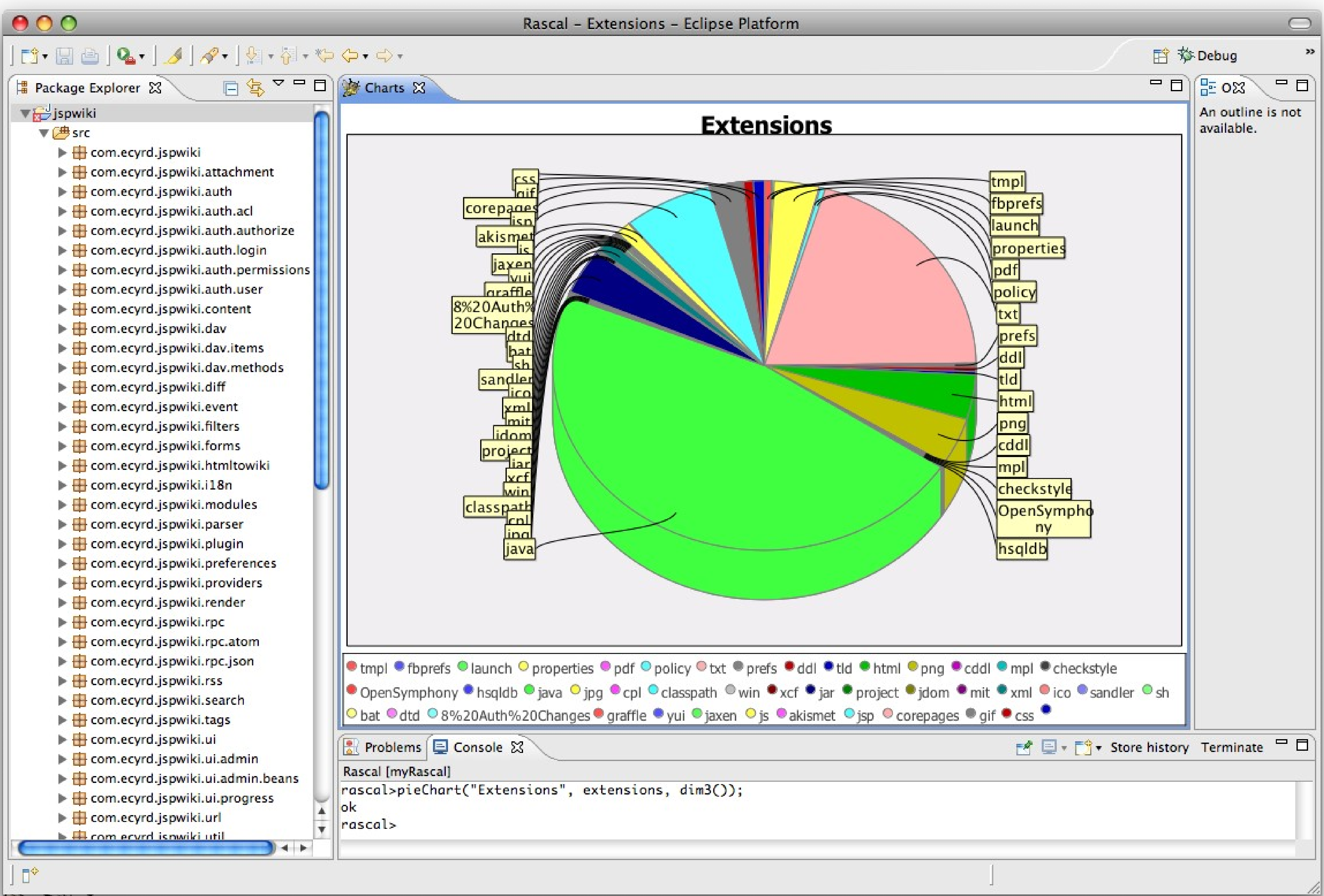
Charting tools

Get all file names from project “jspwiki”

Empty map to count extensions

Visit all filenames and count extensions

Draw them as 3D pieChart


Rascal – Extensions – Eclipse Platform
Debug
Package Explorer
jspwiki
src
com.ecyrd.jspwiki
com.ecyrd.jspwiki.attachment
com.ecyrd.jspwiki.auth
com.ecyrd.jspwiki.auth.acl
com.ecyrd.jspwiki.auth.authorize
com.ecyrd.jspwiki.auth.login
com.ecyrd.jspwiki.auth.permissions
com.ecyrd.jspwiki.auth.user
com.ecyrd.jspwiki.content
com.ecyrd.jspwiki.dav
com.ecyrd.jspwiki.dav.items
com.ecyrd.jspwiki.dav.methods
com.ecyrd.jspwiki.diff
com.ecyrd.jspwiki.event
com.ecyrd.jspwiki.filters
com.ecyrd.jspwiki.forms
com.ecyrd.jspwiki.htmltowiki
com.ecyrd.jspwiki.i18n
com.ecyrd.jspwiki.modules
com.ecyrd.jspwiki.parser
com.ecyrd.jspwiki.plugin
com.ecyrd.jspwiki.preferences
com.ecyrd.jspwiki.providers
com.ecyrd.jspwiki.render
com.ecyrd.jspwiki.rpc
com.ecyrd.jspwiki.rpc.atom
com.ecyrd.jspwiki.rpc.json
com.ecyrd.jspwiki.rss
com.ecyrd.jspwiki.search
com.ecyrd.jspwiki.tags
com.ecyrd.jspwiki.ui
com.ecyrd.jspwiki.ui.admin
com.ecyrd.jspwiki.ui.admin.beans
com.ecyrd.jspwiki.ui.progress
com.ecyrd.jspwiki.url
com.ecyrd.jspwiki.util
Charts
Extensions
css
gif
corepages
jsp
akismet
js
jaxen
yui
graffle
8%20Auth%20Changes
dtd
bat
sh
sandler
ico
xml
mit
jdom
project
jar
xcf
win
classpath
cpl
jpg
java
tmpl
fbprefs
launch
properties
pdf
policy
txt
prefs
ddl
tld
html
png
cddl
mpl
checkstyle
OpenSymphony
hsqldb
tmpl fbprefs launch properties pdf policy txt prefs ddl tld html png cddl mpl checkstyle
OpenSymphony hsqldb java jpg cpl classpath win xcf jar project jdom mit xml ico sandler sh
bat dtd 8%20Auth%20Changes graffle yui jaxen js akismet jsp corepages gif css
An outline is not available.
Problems
Console
Store history
Terminate
Rascal [myRascal]

```
rascal>pieChart("Extensions", extensions, dim3());
ok
rascal>
```

The Rascal Standard Library

- Benchmark
- Boolean
- Exception
- (Labelled) Graph
- Integer
- IO
- JDT (Eclipse only)
- List
- Location
- Map
- Node
- Real
- Relation
- RSF
- Resource (Eclipse only)
- Set
- String
- Subversion
- Tuple
- ValueIO
- viz::Chart
- viz::View (Eclipse only)

Long-term Perspective

- The Rascal language supports the EASY paradigm:
  - creation and execution of fact analysis and transformation tools
  - DSLs
  - meta-programming
- Familiar notation and Eclipse integration lower barrier to entry
- Work in progress

Information

General information:

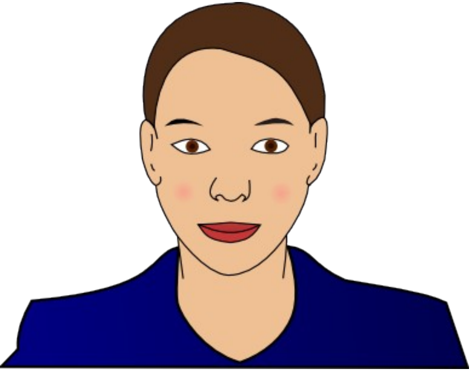
http://www.meta-environment.org

Latest version of Rascal documentation:

http://www.meta-environment.org/doc/books/analysis/rascal-manual/rascal-manual.[html|pdf]

Download Rascal implementation:

http://www.meta-environment.org/Meta-Environment/Rascal



Meet Charlotte

- Charlotte works at a large financial institution in Paris
- Objective: connect legacy software to the web
- Solution:
  - extract call information from the legacy code, analyze it, and synthesize an overview of the call structure
  - Use entry points in the legacy code as entry points for the web interface
  - Automate these transformations

Meet Daniel

- Daniel is concurrency researcher at one of the largest hardware manufacturers worldwide
- Objective: leverage the potential of multi-core processors and find concurrency errors
- Solution:
  - extract concurrency-related facts from the code (e.g., thread creation, locking), analyze these facts and synthesize an abstract automaton
  - Analyze this automaton with third-party verification tools

Meet Elisabeth

- Elisabeth is software architect at an airplane manufacturer
- Objective: Model reliability of controller software
- Solution:
  - describe software architecture with UML and add reliability annotations
  - Extract reliability information and synthesize input for statistics tool
  - Generate executable code that takes reliability into account