Multi-Language Software Analysis with Rascal

Tijs van der Storm
storm@cwi.nl / @tvdstorm

CWI SWAT

Jurgen Vinju (group leader)

reverse engineering, static analysis, renovation

Me

DSLs, language workbenches, language design

Rascal 

- Functional programming with curly braces
- Runs on the JVM
- Command line REPL + Eclipse-based IDE
- Source: https://github.com/usethesource/rascal
- Download: http://www.rascal-mpl.org

Metaprograms

- code visualizers
- refactoring tools
- static analyses
- bug finders
- style checkers
- pretty printers
- smell detectors
- interpreters
- compilers
- metrics tools
- obfuscators
- …

AWK

ANTLR

grep

SQL

etc.

Rascal

http://www.rascal-mpl.org

http://usethesource.io/

Integration

- Data types for
  - concrete syntax trees,
  - abstract syntax trees,
  - source locations,
  - *n*-ary relations
- Pattern matching against all data types
- Comprehensions over all collection types

Finding public fields

- Task: find public fields in Java source code
- Use grep? Imprecise :-(
- Use ANTLR? Too much work :-(
- Use Rascal? Let's see!

Return a list of source locations

The type of Java compilation unit parse trees

```
list[loc] publicFields(start[CompilationUnit] cu)
= [ f@\loc | /(FieldDec)`public <Type _> <Id f>;` := cu ];
```

Search for matching nodes in the tree

Concrete syntax matching, modulo layout

```
start[CompilationUnit] trafoFields(start[CompilationUnit] cu) {
  return innermost visit (cu) {
    case (ClassBody)`{
                    '   <ClassBodyDec* cs1>
                    '   public <Type t> <Id f>;
                    '   <ClassBodyDec* cs2>
                    '}` =>
         (ClassBody)`{
                    '   <ClassBodyDec* cs1>
                    '   private <Type t> <Id f>;
                    '   public <Type t> <Id getter>() {
                    '     return <Id f>;
                    '   }
                    '   public void <Id setter>(<Type t> x) {
                    '     this.<Id f> = x;
                    '   }
                    '   <ClassBodyDec* cs2>
                    '}`
         when
            Id getter := [Id]"get<f>",
            Id setter := [Id]"set<f>"
  }
}
```

Repeat until no more changes

Match source pattern (list matching)

construct new class body

Make getter/setter identifiers

ELSEVIER

Science of Computer Programming

Volume 97, Part 1, 1 January 2015, Pages 143-149

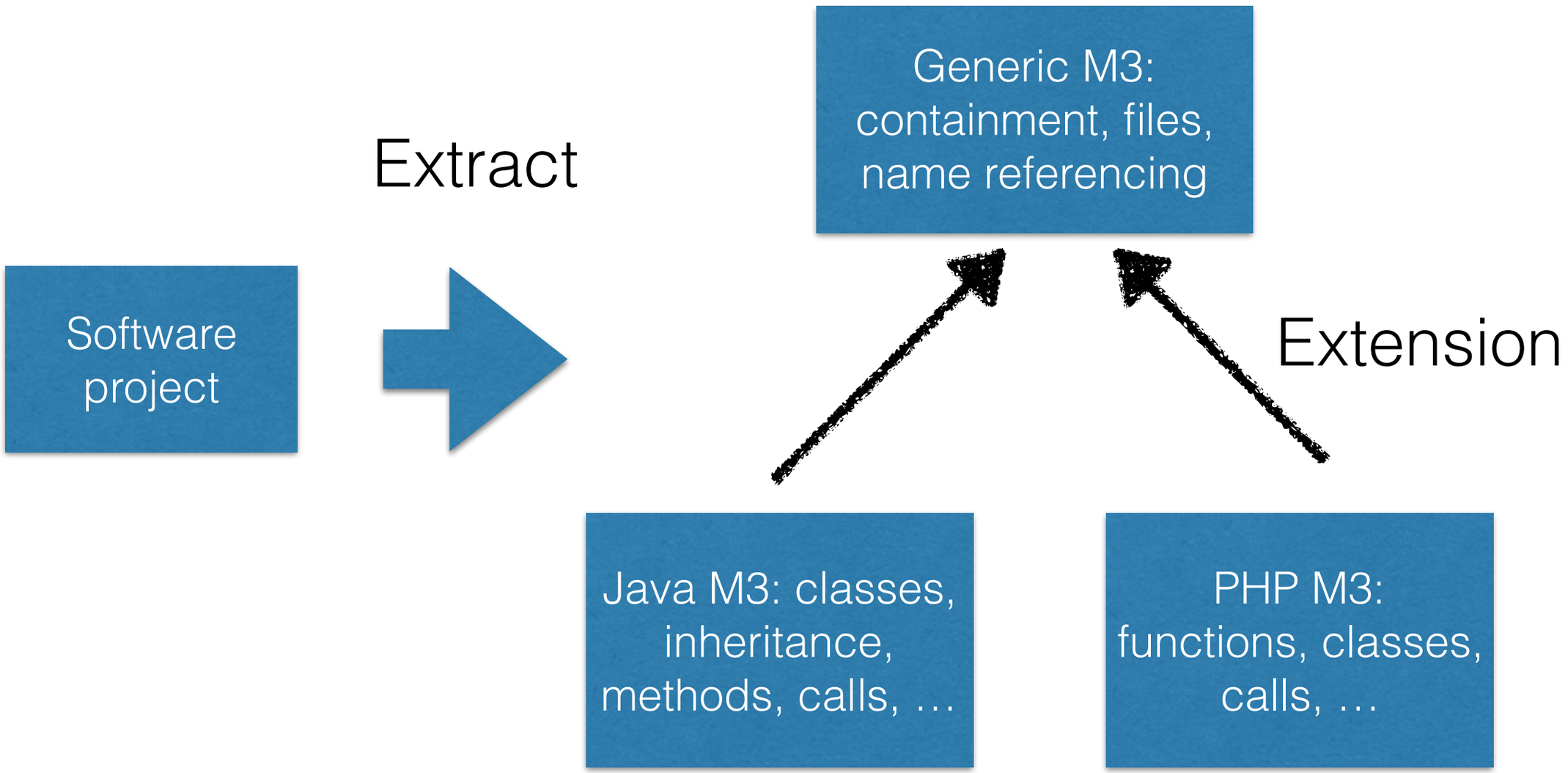
Towards multilingual programming environments

Tijs van der Storm ✉, Jurgen J. Vinju ✉

⊞ Show more

https://doi.org/10.1016/j.scico.2013.11.041

M3: an extensible model for capturing source code facts



Query and synthesize

Generic M3: containment, files, name referencing

Java M3: classes, inheritance, methods, calls, …

PHP M3: functions, classes, calls, …

Analysis results and/or transformations

Core M3
“database schema”

```
data M3(
   rel[loc name, loc src] declarations = {},
   rel[loc name, TypeSymbol typ] types = {},
   rel[loc src, loc name] uses = {},
   rel[loc from, loc to] containment = {},
   list[Message] messages = □,
   rel[str simpleName, loc qualifiedName] names = {},
   rel[loc definition, loc comments] documentation = {},
   rel[loc definition, Modifier modifier] modifiers = {}
) = m3(loc id);
```

The source location

scheme

Authority

Path

```
|project://rascal-ecore/src/lang/ecore/
Refs.rsc|(1821,130,<54,0>,<56,1>))
```

File offset

Length

begin and end column and line

Logical locations

- |java+field://java/lang/System/out|
- |java+method://java/lang/System/out.println(Object)|
- ...

logical

physical

```
rel[loc name, loc src] declarations = {},
```

physical

logical

```
rel[loc src, loc name] uses = {}
```

Simple example: JStm

- State machine DSL with integrated Java
- Compiles to plain Java class
- Create custom M3 for DSL
- Merge with “stock” M3 for Java
- => cross language analysis ;)

```
package doors;

import java.util.List;
import java.util.ArrayList;

statemachine Doors {
  private List<String> tokens = new ArrayList<String>();

  event open "OP2K";
  event close "CL2K";

  state closed {
    System.out.println("We're closed now");
    tokens.add(token);
    on open ⇒ opened;
  }

  state opened {
    System.out.println("We're opened now");
    on close ⇒ closed;
  }

}
```

```
package doors;

import java.util.List;
import java.util.ArrayList;

statemachine Doors {
  private List<String> tokens = new ArrayList<String>();

  event open "OP2K";
  event close "CL2K";

  state closed {
    System.out.println("We're closed now");
    tokens.add(token);
    on open ⇒ opened;
  }

  state opened {
    System.out.println("We're opened now");
    on close ⇒ closed;
  }

}
```

Java Code

Java Code

Java Code

Java Code

```
package doors;

import java.util.List;
import java.util.ArrayList;

statemachine Doors {
  private List<String> tokens = new ArrayList<String>();

  event open "OP2K";
  event close "CL2K";

  state closed {
    System.out.println("We're closed now");
    tokens.add(token);
    on open ⇒ opened;
  }

  state opened {
    System.out.println("We're opened now");
    on close ⇒ closed;
  }

}
```

DSL code

DSL code

DSL code

JStm Doors

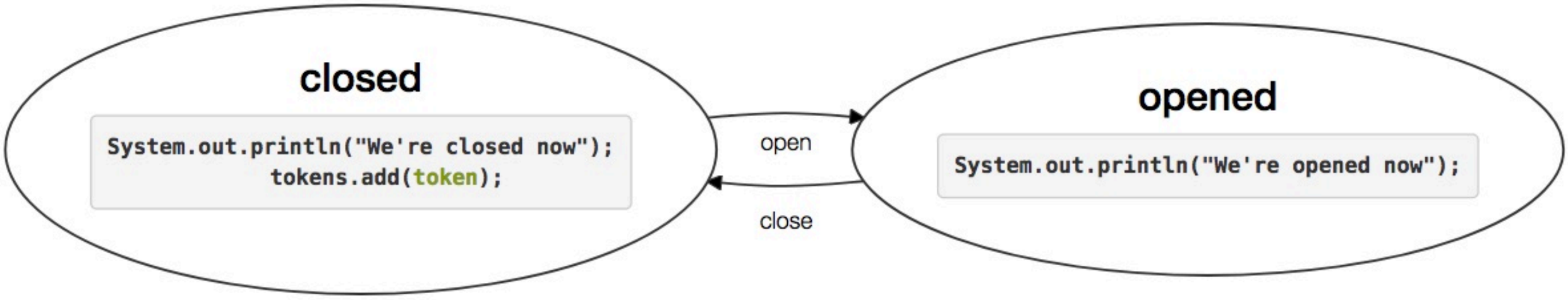
Code

```
private List<String> tokens = new ArrayList<String>();
```

Events

EventToken
open "OP2K"
close "CL2K"

State machine



Analysis questions

- Back linking: which state does this Java code belong to?
- Reachability: which Java methods are reachable from processing event token “x”?
- Type checking embedded Java code
- Name resolution across language boundaries
- Rename state machine => rename in Java client code
- …

dsl — A domain specific language, where code is written in one language and errors are given in another.

https://programmingisterrible.com/post/65781074112/devils-dictionary-of-programming


LIVE DEMO

Summary

- Meta programming with **Rascal**: from ad hoc to systematic
- **M3**: a generic source code model
- Entities identified by (logical) **source locations**
- **Cross language** linking of entities
- Example: **JStm** language => DSL + Java