Software Analysis and Transformation with Rascal

BioAssist Meeting
Jan 11th, 2013
Jurgen Vinju
• Centrum Wiskunde & Informatica
• Programming languages and systems
  • Algol
  • Python
  • ASF+SDF, Rascal
  • MonetDB
• Where mathematics meets informatics
  • striving for fundamental (general) results
  • motivated by and applied in industry, government, and the sciences
• W3C
• Software Improvement Group (spin-off)
• Master Software Engineering @{Universiteit van Amsterdam, VU, HvA}
25 minutes

- What and why do we research software at CWI?
- How?
- Two possible discussions
  - Question: how is bio software unique?
  - Perspective: meta = data programming?
Our team

Paul Klint
Jurgen Vinju
Tijs v/d Storm
Bob Fuhrer
The problem with software is not in constructing it

(given sufficiently experienced architects & engineers)
The problem is in understanding existing software in order to improve it

(and a lot of software exists)
We study software systems: their design, their construction and their inevitable evolution.

- learning to understand software systems
- learning to improve them
- focusing on complexity as the quality attribute
- studying the causes of software complexity
- studying solutions to get simpler software
- helping software engineers to be more effective
- shaping the future of programming languages and IDEs

(NASA mission control, apollo 13)
Software is not so difficult to understand, but it is extremely complex.
Software - large and complex structures of computer instructions, written and read by man, executed by computers.

“marked by a senseless, disorienting, often menacing complexity...” (Infoplease.com)
The source code of "ls"

3894 lines

367 ifs

174 cases
Solution...

Tools
Transformation & Analysis

- (de)optimization
- GOTO removal
- Bug fixing (Y2K)
- Porting
- Refactoring ...
- Model-to-code
- Languages

Raphael (1509)

- Code-to-model
- Quality assessment
- Mining trends
- Dead code detection
- Bug detection
- Model checking
- Impact analysis

(etc)
“every week a new tool”

Research

Tools

Application
Rascal is a DSL for meta programming.
The three challenges

Diversity

Multi-disciplinary

Precision vs Efficiency

Friday, January 11, 13
The key point of Rascal is that it is a one-stop-shop; no hacking stuff together, just one consistent, typed, and safe environment for meta-programming for “any” language.
D.I.Y.

- That’s the goal
- We teach Rascal (master)
- We use Rascal
- Caveat: “Experimental”
A one-slide DSL
(or a one-slide DSL)

```javascript
metro {
    Centraal Waterloo Weesperplein Wibautstraat Amstel;
    Amstel Spaklerweg Overamstel Rai Zuid;
    Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
    Centraal Rokin FerdinandBol Zuid;
}
```
A one-slide DSL

```javascript
metro {
  Centraal Waterloo Weesperplein Wibautstraat Amstel;
  Amstel Spaklerweg Overamstel Rai Zuid;
  Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
  Centraal Rokin FerdinandBol Zuid;
}

{ "Centraal", "Waterloo" },
```
A one-slide DSL

```json
metro {
  Centraal Waterloo Weesperplein Wibautstraat Amstel;
  Amstel Spaklerweg Overamstel Rai Zuid;
  Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
  Centraal Rokin FerdinandBol Zuid;
}

{ "Centraal", "Waterloo"},
{ "Waterloo"," Weesperplein"}, ...
```
A one-slide DSL

```plaintext
metro {
  Centraal Waterloo Weesperplein Wibautstraat Amstel;
  Amstel Spaklerweg Overamstel Rai Zuid;
  Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
  Centraal Rokin FerdinandBol Zuid;
}

{ <“Centraal”, “Waterloo”>,
  <“Waterloo”, “Weesperplein”>, … }

digraph Metro {
  node [shape=box]
  Centraal -> Waterloo
  Waterloo -> Weesperplein ...
  Centraal [shape=ellipse]
}
```
A one-slide DSL
module Metro
module Metro

start syntax System = "metro" "{" Track* tracks "}";
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\ \t\n\r]*;
module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {
        <from, to> | /Track t := parse(#start[System], source),
    }
module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =

  {<from, to> | /Track t := parse(#start[System], source),
     (Track) `\<Id+ _\> <Id from> <Id to> <Id+ _\>;` := t};
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | //Track t := parse(#start[System], source),
    (Track) `\Id+ _` \Id from\Id to\Id+ _`; \t
  };
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
   {<from, to> | /Track t := parse(#start[System], source),
     (Track) `Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;
void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {

module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [
\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =

{<from, to> | /Track t := parse(#start[System], source),
(Track) `<Id+ _> <Id from> <Id to> <Id+ _>;<` := t};

bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro { node [shape=box]
module Metro

start syntax System = "metro" "{{ Track* tracks }}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
   {<from, to> | /Track t := parse(#start[System], source),
      (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};

bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro { node [shape=box]
        <for (<from, to> <- metro) {

Friday, January 11, 13
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [
\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ `_> <Id from> <Id to> <Id+ `_>;`:` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro {
        node [shape=box]
            `<for (<from, to> <- metro) {
                ` `<from> -\> <to>`

Code
Model
Picture
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
   {<from, to> | Track t := parse(#start[System], source),
      (Track) `<Id+ `_> <Id from> <Id to> <Id+ `_>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
   writeFile(target,"digraph Metro { node [shape=box]
      `<for (<from, to> <- metro) {
         ` <from> `\> <to>`
      `<for (st <- metro<from>, isHub(metro, st))`
A one-slide DSL

module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
   {<from, to> | /Track t := parse(#start[System], source),
      (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
   writeFile(target,"digraph Metro { node [shape=box]
      `<for (<from, to> <- metro) {>
         `<from> -\> <to></>
      `<for (st <- metro<from>, isHub(metro, st)){>
         `<st> [shape=ellipse]<}
   
Friday, January 11, 13
module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =

  {<from, to> | /Track t := parse(#start[System], source),
      (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};

bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,"digraph Metro { node [shape=box]
      "<for (from, to) <- metro) {
        " <from> -\> <to></>
      "<for (st <- metro<from>, isHub(metro, st)){
        " <st> [shape=ellipse]<}
      "}",


module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [
\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;
void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro { node [shape=box]
        ‘<for (<from, to> <- metro) {>
            ‘ <from> -\> <to><}
        ‘<for (st <- metro<from>, isHub(metro, st)){>
            ‘ <st> [shape=ellipse]<}
        ‘}”);}
A one-slide DSL
A one-slide DSL

What is the point?

- Rapid tool development
- No boilerplate
- No glue
- No magic
- Done. Next!
A one-slide DSL

What is the point?
- Rapid tool development
- No boilerplate
- No glue
- No magic
- Done. Next!

This works for
- all kinds of meta-programming tools
- all kinds of languages
Library development
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

- **Front-ends** for programming languages
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices
- **Front-ends** for programming languages
- **Generic analyses**; statistics, constraints, satisfiability, …
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices
- **Front-ends** for programming languages

- **Generic analyses**: statistics, constraints, satisfiability, …

- **Visualization**: one-stop-library for any visualization (graph, chart, browser, …)
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

- **Front-ends** for programming languages

- **Generic analyses**: statistics, constraints, satisfiability, …

- **Visualization**: one-stop-library for any visualization (graph, chart, browser, …)

- *(this is our main challenge at the moment)*
Current applications

- PHP, Lua static analysis of dynamic languages
- Modular/Language parametric refactoring
- Grammar engineering
- Domain specific languages
  - Pacioli - Computational auditing
  - Derric - Digital Forensics
- Design pattern diagnostics
Take home messages

- http://www.rascal-mpl.org
- for DIY tool building
- open-source
- CWI - SWAT
  - studies real software (for example yours)
  - builds tools
- UvA Master Software Engineering
  - part-time (2 year), full-time (1 year)
  - (to be developed) “deep track” - domain specific SE tracks
Discussion

• For meta programming source code is data
• and, source code is big!
• so, is meta programming like big data?
• Common challenges, common solution patterns?
• Are meta programming solutions relevant for bd?
• Are big data solutions relevant for meta prog?
• {C,sh,w}ould Rascal be extended to big data use cases?