Generating VScode extensions for **DSLs** using **Rascal**

Jurgen Vinju  
*Strumenta Online Community*  
March 3rd, 2022
In no particular order, many people have contributed in one way or another to the software that I’m about to demonstrate:

Paul Klint, Tijs van der Storm, Mark van den Brand, Mark Hills, Frank Tip, Arie van Deursen, Michael Steindorfer, Davy Landman, Hayco de Jong, Pieter Olivier, Merijn de Jonge, Jeroen Scheerder, Eelco Visser, Joost Visser, Ali Afrozeh, Anastasia Izmaylova, Bastiaan Heeren, Alexander Serebrenik, Rodin Aarssen, Riemer van Rozen, Aiko Yamashita, Anya Helene Bagge, Arnold Lankamp, Ashim Shahi, Bas Basten, Bert Lisser, Vadim Zaystev, Jasper Timmer, Jouke Stoel, Jeroen van den Bos, Lina Maria Ochoa Venegas, Mauricio Verano Merino, Pablo Inostroza Valdera, Thomas Degueule, Ralf Lämmel, Dinesh, Jan Rekers, Emma van der Meulen, Wilco Koorn, Susan Uskudarli, Tobias Kuipers, Leon Moonen, Jasper Kamperman, Magiel Bruntink, Thomas van Binsbergen, Chris Verhoef, Alex Sellink, Tim Soethout, ... (...apologies if you should have been on this list)
“Generating IDEs”

- **Generic components**
  - **Language Workbench**
    - **(Domain-specific) Language Definition**
      - meta code
    - **(Domain-specific) code/models**
      - data

- **IDE**
  - user
  - meta programmer
  - programmer

- **Software**
Agenda

- History
- Design
- Tutorial
- Live Demo
- Pointers
- Questions
History of Rascal

- ASF+SDF + LeLISP          1983 - 1998
- ASF+SDF Meta-Environment  1999 - 2008
- ASF+SDF Meta + IMP + Eclipse 2007 - 2008
- Rascal + IMP + Eclipse    2009 - ...
- Rascal + LSP + VScode     2021 - ...
ASF+SDF+LeLISP (1984-1998)

• EEC programs for research and development in information technologies:
• GIPE: “Generating Interactive Programming Environments”
• LeLISP: LISP + homemade GUI on SUN Sparc/Solaris
• LISP macros, LISP libraries, everything LISP
• SDF: Syntax Definition Formalism
• ASF: Algebraic Specification Formalism
• ASF+SDF Concrete Syntax: tree patterns and expressions in object language syntax (!!!)
• BOX: declarative pretty printing formalism
• SEAL: declaratively linking (generated) language processors to the IDE (buttons, etc.)
• ±15 related Ph.D. theses (Paul Klint et al.), much (inter)national collaboration
• COBOL software renovation applications (Chris Verhoef et al.)
• In 1994 —I was 17 years old— Paul Klint visited my high school and demonstrated this system
ASF+SDF Meta-Environment (1999-2008)

- Re-imagining of ASF+SDF (Mark van den Brand et al.) in C, Java, ASF+SDF, ToolBus
- **ToolBus** (1998, Paul Klint, Jan Bergstra)
  - Strict separation of coordination from computation
  - Connecting tools with sockets and protocols: “LSP” avant-la-lettre (!)
  - Process Algebra (ACP): DSL for coordinating (generated) tools
- Drop-in replacement of editors (Java, (g)vim, (x)emacs, Eclipse)
- SDF2 - scannerless generalized LR parsing (Eelco Visser et al.)
- ASF compiler to C - *very* fast term rewriting (Mark van den Brand et al.)
- TIDE - generic/language parametric debugging (Pieter Olivier)
- RScript - Relational Calculus DSL for analysis (Paul Klint)
- APIgen - algebraic data-types for C and Java (De Jong, Olivier, Vinju)
- ATerms - *very* fast (serialized) terms + GC (Pieter Olivier, Hayco de Jong)
- Traversal Functions - structure-shy recursive functions (Vinju et al.)
- Many applications: COBOL, C, Java, Action Semantics (Peter Mosses)
ASF+SDF Meta + IMP + Eclipse (2007 - 2008)

- Eclipse IMP “IDE Meta-tooling Platform” (Robert Fuhrer et al.) - Java, Eclipse Extension points: parametric interface for programming languages [OOPSLA 2008]
- IMP to ToolBus: every callback is a tool
- “ASF+SDF in Eclipse”
- Immediate Language Workbench; no “second level”
- More varied standard interactions with language semantics:
  - highlighters, outlines, search, jump-to-def, find references, completion
Rascal’s origins (2008)

- ASF, SDF, Box, Meta-Environment, RScript, APIgen, ATerms, ToolBus:
  - separation-of-concerns: a “meta-DSL” for every aspect of a meta program
  - integration was a big challenge for us (and for the meta-programmers)
  - meta-programmer context-switching between meta-languages for the same object language
  - data marshalling overhead between different language processors

- World was changing:
  - many new students without formal methods backgrounds
  - Eclipse was becoming the de-facto standard for language development
  - Other transformation languages like TXL, StrategoXT, TOM, DMS started using external database technology
  - Other reverse engineering frameworks based on relations and their visualizations (Grok, Crokopat, Rigi)

- Rascal design requirements:
  - Expressive: Cover all the of power of ASF2, SDF2, Box, RScript, ApiGen and ATerms
  - Learnable: understandable without formal methods background
  - Integrated: no more context-switching and no more data marshalling
  - Hi-Fi: origin tracking to source locations, no a priori loss of comments and whitespace
  - Interactive: (Loosely) integrated with Eclipse
Rascal + IMP + Eclipse (2009)

• IMP provides an “LSP”
• Rascal generates implementations for IMP services (parser, outliner, checker, etc.)

• Rascal URI for Eclipse [project://.../

• Immediate Language Workbench (no 2nd level)
• Powerful Read-Eval-Print loop
• Integrated web visualizations
• Integrated interactive
Rascal + LSP + VScode (2021)

- The world changed again; now VScode is the defacto standard for code editing. Plus SWAT.engineering (2017) has interest.

- Microsoft Language Server Protocol is very much like ToolBus and IMP but with JSON-RPC instead of ATerms or Java services.

- Yet another “port” of the Meta-Environment, no conceptual difference

- Beyond the LSP:
  - terminals for REPLs
  - web servers plus web clients for interactive visualizations
  - Immediate language workbench: first level extensions
Rascal Design (2009)

Syntax Definition

Procedural & Functional Programming

Term Rewriting

Relational Calculus

EASY: Extract, Analyze, Synthesize

code

syntax trees

relations

parsing

unparsing

templating

rewriting

fact extraction

synthesis

analysis, inference

SWAT - Software Analysis And Transformation
Rascal is a nominally-typed procedural programming language with familiar keywords like **if**, **for**, **while** and **return**, and some extras like **visit**, case patterns.
**Tutorial**

```c
int fac(0) = 1;
default int fac(int n) = n * fac(n-1);
```

Pattern matching is everywhere!
data Num = zero();
data Num = succ(Num pred);
Num fac(zero()) = succ(zero());
Num fac(s:succ(n)) = mult(s,fac(n));

(abstract) syntax is everwhere in Rascal
**Tutorial**

**lexical**  
Id = [a-z]+;

**syntax**  
Exp = Id id  
| (“(“ Exp “)”)  
> Exp lhs “*” Exp rhs  
> Exp lhs “+” Exp rhs;

Exp e = [Exp] “1 + 2 * 3”;  
e = e.rhs;

Exp simplify((Exp) `<Exp a> * (<Exp b> + <Exp c>)`)  
= (Exp) `<Exp a> * <Exp b> + <Exp a> * <Exp c>`;
Tutorial

// import a syntax definition

import lang::myLanguage::Syntax;

// define a location:
loc example = |project://demo/examples/fac.program|;

// parse the contents of the location
Program cu = parse(#Program, example);

// traverse the tree, find all assignments, create a binary relation
rel[str name, loc place]

assigned = { <"<id>".src> | /(Stat) `<Id id> = <Exp e>` : cu };
declared = …
declaredButNeverAssigned = declared.name - assigned.name;

warnings = {warning("<n> was declared but never assigned.", p) | <n,p> ← declared, n in declaredButNeverAssigned};
module util::LanguageServer

alias Parser = Tree (str input, loc origin);
alias Summarizer = Summary (loc origin, Tree input);
alias Outliner = list[DocumentSymbol] (Tree input);
alias Completer = list[Completion] (Tree input, str prefix, int offset);
alias Builder = list[Message] (list[loc] sources, PathConfig pcfg);
alias LensDetector = rel[loc src, Command lens] (Tree input);
alias CommandExecutor = void (Command command);
alias InlayHinter = list[InlayHint] (Tree input);

data LanguageService = parser(Parser parser) | summarizer(Summarizer summarizer) | ...
data Language = language(PathConfig pcfg, str name, str ext, str module, str contributor);

module MyLanguage

set[LanguageService] myServices() { return ...; }

void main() {
  registerLanguage(language(pcfg, “MyLanguage”, “ml”, “Main”, “myServices”));
}

Tutorial: LSP in Rascal
Tutorial: IDE services

module util::IDEServices // accessing Eclipse or VScode features from LSP callbacks (asynchronously)

java void browse(loc uri);  // uses WebView in VScode
java void edit(loc uri);    // opens editor in VScode
java void applyDocumentsEdits(list[DocumentEdit] edits); // applies edits (with undo stack and/or preview)
java void showInteractiveContent(Content content); // serves interactive content in WebView window
java void showMessage(Message message);            // pop-up
java void logMessage(Message message);            // via log4j and logging API in LSP
java void registerDiagnostics(list[Message] messages); // Problems view
java void unregisterDiagnostics(list[loc] resources);
java void registerLanguage(Language lang); // adds a new language to the IDE
Dataflow

1. registerLanguage

Rascal

registered DSL callbacks

IDE services API

language processing

VScod

 Terminals

REPLs

parametric LSP server

Dispatch on file extension

Data marshalling

10

9

8

7

4

12

5

13

2

10

11

11

6
Demo

• Very small language
• Simple intermediate model
• Live programming
• IDE with
  • syntax highlighting,
  • outline,
  • checker,
  • commands
Selected Rascal Libraries

- analysis::text::search - Lucene based indexing/search
- analysis::flow - parametric dataflow analysis
- lang::java - JDT-based Java analysis in Rascal
- lang::box - Box-based string formatting DSL
- TypePal - declarative name/type analysis/inference
- FlyBytes - JVM bytecode analysis/generation
- CLAiR - C++ language analysis in Rascal
- PHP-air - PHP language analysis in Rascal
- Python-air - Python language analysis in Rascal
- Lua-air - LUA language analysis in Rascal
- JS-air - JavaScript language analysis in Rascal
- AlleAlle - relation model find with theories (parametric model checker)
- Bacatá - Jupyter Notebook Generator
- Rascal.ecore - live (bidirectional) rascal <-> ecore
- Maracas - Java API breaking change analysis and breaking change impact analysis
- Salix - Elm-style interactive GUIs in Rascal
Example DSLs

- Rebel2 - state-machine based scalable transaction systems (IDE, simulator, model-checking, code-generation)
- Derric - (Binary) File Formats (IDE, debugger, code-generation, file recovery algorithms)
- LiveQL - DSL for questionnaires
- Jeff - OO language with effect handling
- Marvol - DSL to make the NOA robot dance
- Amalga - mini-language for image algorithm expression
- (e)Flint - DSL for the expression of norms (laws)
- Bird - (Binary) File Formats
- Nescio - DSL for anonymization of binary data (coupled to Bird)
- ScriptButler - an IDE and game engine for puzzle game design
- Ludoscope Mini - a textual DSL and engine for level design.
Further Pointers

- [http://www.rascal-mpl.org](http://www.rascal-mpl.org) - Rascal homepage
- [http://github.com/usethesource](http://github.com/usethesource) - Github organization for Rascal
- [https://stackoverflow.com/questions/tagged/rascal](https://stackoverflow.com/questions/tagged/rascal) - >400 StackOverflow answers
- [https://github.com/cwi-swat](https://github.com/cwi-swat) - lots of experimental Rascal libraries and apps
- [https://www.swat.engineering](https://www.swat.engineering) - language engineering services
  (look out for job opportunities)

@JurgenVinju Jurgen.Vinju@cwi.nl