IN PARTNERSHIP WITH:
Centrum Wiskunde & Informatica

Activity Report 2015

Project-Team ATEAMS

Analysis and Transformation based on rEliAble tool coMpositionS

RESEARCH CENTER
Lille - Nord Europe

THEME
Architecture, Languages and Compilation

Table of contents

Project-Team ATEAMS

Creation of the Project-Team: 2009 July 01, end of the Project-Team: 2015 December 31

Keywords:

Computer Science and Digital Science:

1. - Architectures, systems and networks
2. - Software
2.1.1. - Semantics of programming languages
2.1.10. - Domain-specific languages
2.1.2. - Object-oriented programming
2.1.3. - Functional programming
2.2.1. - Static analysis
2.5. - Software engineering

Other Research Topics and Application Domains:

6.1. - Software industry
6.1.1. - Software engineering
6.1.2. - Software evolution, maintenance
6.6. - Embedded systems

1. Members

Research Scientists

Jurgen Vinju [Team leader, Centrum Wiskunde & Informatica, Professor]
Tijs Van Der Storm [Centrum Wiskunde & Informatica]
Jan Van Eijck [Centrum Wiskunde & Informatica]

Faculty Member

Paul Klint [Centrum Wiskunde & Informatica, Professor]

Engineers

Maarten Dijkema [Centrum Wiskunde & Informatica]
Bert Lisser [Centrum Wiskunde & Informatica]

PhD Students

Ali Afroozeh [Centrum Wiskunde & Informatica]
Pablo Inostroza Valdera [Centrum Wiskunde & Informatica]
Anastasia Izmaylova [Centrum Wiskunde & Informatica]
Davy Landman [Centrum Wiskunde & Informatica]
Michael Steindorfer [Centrum Wiskunde & Informatica]
Riemer Van Rozen [Centrum Wiskunde & Informatica]
Jouke Stoel [Centrum Wiskunde & Informatica]

Administrative Assistant

Sandrine Meilen [Inria]

2. Overall Objectives

2.1. Presentation

Software is very complex, and it seems to become more complex every year. Over the last decades, computer science has delivered various insights how to organize software better. Via structured programming, modules, objects, components and agents, these days software systems are more and more evolving into "systems of systems" that provide services to each other. Each system is large, uses incompatible — new, outdated or non-standard — technology and above all, exhibits failures.

It is becoming more and more urgent to analyze the properties of these complicated, heterogeneous and very large software systems and to refactor and transform them to make them simpler and to keep them up-to-date. With the plethora of different languages and technology platforms it is becoming very difficult and very expensive to construct tools to achieve this.

The main challenge of ATEAMS is to address this combination of a need for all kinds of novel analysis and transformation tools and the existence of the diversity of programming environments. We do this by investing in a virtual laboratory called "Rascal". It is a domain specific programming language for source code analysis, transformation and generation. Rascal is programming language parametric, such that it can be used to analyze, transform or generated source code in any language. By combining concepts from both program analysis and transformation into this language we can efficiently experiment with all kinds of tools and algorithms.

We now focus on three sub-problems. Firstly, we study software analysis: to extract information from existing software systems and to analyze it. The extracted information is vital to construct sound abstract models that can be used in further analysis. We apply these extraction techniques now to analyze (large bodies of) source code: finding bugs and finding the causes of software complexity.

Secondly, we study refactoring: to semi-automatically improve the quality of a software system without changing its behavior. Refactoring tools are a combination of analysis and transformations. Implementations of refactoring tools are complex and often broken. We study better ways of designing refactorings and we study ways to enable new (more advanced and useful) refactorings. We apply these refactorings now to isolate design choices in large software systems and compare systems that are equal except a single design choice.

Finally, we study code generation from domain specific languages (DSLs). Here we also find a combination of analysis and transformation. Designing, implementing and, very importantly, maintaining DSLs is costly. We focus on application areas such as Computational Auditing, Game Economies, and Core Banking to experiment with this subject. In Computational Auditing we are focusing on modeling interactive questionnaires. The Game economies domain involves modeling and verifying the dynamic behaviour of game play. Core banking requires the formal modeling of financial services and products.

3. Research Program

3.1. Research method

We are inspired by formal methods and logic to construct new tools for software analysis, transformation and generation. We try and proof the correctness of new algorithms using any means necessary.

Nevertheless we mainly focus on the study of existing (large) software artifacts to validate the effectiveness of new tools. We apply the scientific method. To (in)validate our hypothesis we often use detailed manual source code analysis, or we use software metrics, and we have started to use more human subjects (programmers).

Note that we maintain ties with the CWI spinoff "Software Improvement Group" which services most of the Dutch software industry and government and many European companies as well. This provides access to software systems and information about software systems that is valuable in our research.

3.2. Software analysis

This research focuses on source code; to analyze it, transform it and generate it. Each analysis or transformation begins with fact extraction. After that we may analyze specific software systems or large bodies of software systems. Our goal is to improve software systems by understanding and resolving the causes of software complexity. The approach is captured in the EASY acronym: Extract Analyze SYnthesize. The first step is to extract facts from source code. These facts are then enriched and refined in an analysis phase. Finally the result is synthesized in the form of transformed or generated source code, a metrics report, a visualization or some other output artifact.

The mother and father of fact extraction techniques are probably Lex, a scanner generator, and AWK, a language intended for fact extraction from textual records and report generation. Lex is intended to read a file character-by-character and produce output when certain regular expressions (for identifiers, floating point constants, keywords) are recognized. AWK reads its input line-by-line and regular expression matches are applied to each line to extract facts. User-defined actions (in particular print statements) can be associated with each successful match. This approach based on regular expressions is in wide use for solving many problems such as data collection, data mining, fact extraction, consistency checking, and system administration. This same approach is used in languages like Perl, Python, and Ruby. Murphy and Notkin have specialized the AWK-approach for the domain of fact extraction from source code. The key idea is to extend the expressivity of regular expressions by adding context information, in such a way that, for instance, the begin and end of a procedure declaration can be recognized. This approach has, for instance, been used for call graph extraction but becomes cumbersome when more complex context information has to be taken into account such as scope information, variable qualification, or nested language constructs. This suggests using grammar-based approaches as will be pursued in the proposed project. Another line of research is the explicit instrumentation of existing compilers with fact extraction capabilities. Examples are: the GNU C compiler GCC, the CPPX C++ compiler, and the Columbus C/C++ analysis framework. The Rigi system provides several fixed fact extractors for a number of languages. The extracted facts are represented as tuples (see below). The CodeSurfer source code analysis tool extracts a standard collection of facts that can be further analyzed with built-in tools or user-defined programs written in Scheme. In all these cases the programming language as well as the set of extracted facts are fixed thus limiting the range of problems that can be solved.

The approach we are exploring is the use of syntax-related program patterns for fact extraction. An early proposal for such a pattern-based approach consisted of extending a fixed base language (either C or PL/1 variant) with pattern matching primitives. In our own previous work on RScript we have already proposed a query algebra to express direct queries on the syntax tree. It also allows the querying of information that is attached to the syntax tree via annotations. A unifying view is to consider the syntax tree itself as “facts” and to represent it as a relation. This idea is already quite old. For instance, Linton proposes to represent all syntactic as well as semantic aspects of a program as relations and to use SQL to query them. Due to the lack of expressiveness of SQL (notably the lack of transitive closure) and the performance problems encountered, this approach has not seen wider use.

Parsing is a fundamental tool for fact extraction for source code. Our group has longstanding contributions in the field of Generalized LR parsing and Scannerless parsing. Such generalized parsing techniques enable generation of parsers for a wide range of existing (legacy) programming languages, which is highly relevant for experimental research and validation.

Extracted facts are often refined, enriched and queried in the analysis phase. We propose to use a relational formalization of the facts. That is, facts are represented as sets of tuples, which can then be queried using relational algebra operators (e.g., domain, transitive closure, projection, composition etc.). This relational representation facilitates dealing with graphs, which are commonly needed during program analysis, for instance when processing control-flow or data-flow graphs. The Rascal language integrates a relational sublanguage by providing comprehensions over different kinds of data types, in combination with powerful pattern matching and built-in primitives for computing (transitive/reflexive) closures and fixpoint computations (equation solving).

3.2.1. Goals

The main goal is to replace labour-intensive manual programming of fact extractors by automatic generation based on concise and formal specification. There is a wide open scientific challenge here: to create a uniform and generic framework for fact extraction that is superior to current more ad-hoc approaches, yet flexible enough to be customized to the analysis case at hand. We expect to develop new ideas and techniques for generic (language-parametric) fact extraction from source code and other software artifacts.

Given the advances made in fact extraction we are starting to apply our techniques to observe source code and analyze it in detail.

3.3. Refactoring and Transformation

The second goal, to be able to safely refactor or transform source code can be realized in strong collaboration with extraction and analysis.

Software refactoring is usually understood as changing software with the purpose of increasing its readability and maintainability rather than changing its external behavior. Refactoring is an essential tool in all agile software engineering methodologies. Refactoring is usually supported by an interactive refactoring tool and consists of the following steps:

- Select a code fragment to refactor.
- Select a refactoring to apply to it.
- Optionally, provide extra parameter needed by the refactoring (e.g., a new name in a renaming).

The refactoring tool will now test whether the preconditions for the refactoring are satisfied. Note that this requires fact extraction from the source code. If this fails the user is informed. The refactoring tool shows the effects of the refactoring before effectuating them. This gives the user the opportunity to disable the refactoring in specific cases.The refactoring tool applies the refactoring for all enabled cases. Note that this implies a transformation of the source code. Some refactorings can be applied to any programming language (e.g., rename) and others are language specific (e.g., Pull Up Method). At http://www.refactoring.com an extensive list of refactorings can be found.

There is hardly any general and pragmatic theory for refactoring, since each refactoring requires different static analysis techniques to be able to check the preconditions. Full blown semantic specification of programming languages have turned out to be infeasible, let alone easily adaptable to small changes in language semantics. On the other hand, each refactoring is an instance of the extract, analyze and transform paradigm. Software transformation regards more general changes such as adding functionality and improving non-functional properties like performance and reliability. It also includes transformation from/to the same language (source-to-source translation) and transformation between different languages (conversion, translation). The underlying techniques for refactoring and transformation are mostly the same. We base our source code transformation techniques on the classical concept of term rewriting, or aspects thereof. It offers simple but powerful pattern matching and pattern construction features (list matching, AC Matching), and type-safe heterogenous data-structure traversal methods that are certainly applicable for source code transformation.

3.3.1. Goals

Our goal is to integrate the techniques from program transformation completely with relational queries. Refactoring and transformation form the Achilles Heel of any effort to change and improve software. Our innovation is in the strict language-parametric approach that may yield a library of generic analyses and transformations that can be reused across a wide range of programming and application languages. The challenge is to make this approach scale to large bodies of source code and rapid response times for precondition checking.

3.4. The Rascal Meta-programming language

The Rascal Domain-Specific Language for Source code analysis and Transformation is developed by ATeams. It is a language specifically designed for any kind of meta programming.

Meta programming is a large and diverse area both conceptually and technologically. There are plentiful libraries, tools and languages available but integrated facilities that combine both source code analysis and source code transformation are scarce. Both domains depend on a wide range of concepts such as grammars and parsing, abstract syntax trees, pattern matching, generalized tree traversal, constraint solving, type inference, high fidelity transformations, slicing, abstract interpretation, model checking, and abstract state machines. Examples of tools that implement some of these concepts are ANTLR, ASF+SDF, CodeSurfer, Crocopat, DMS, Grok, Stratego, TOM and TXL. These tools either specialize in analysis or in transformation, but not in both. As a result, combinations of analysis and transformation tools are used to get the job done. For instance, ASF+SDF relies on RScript for querying and TXL interfaces with databases or query tools. In other approaches, analysis and transformation are implemented from scratch, as done in the Eclipse JDT. The TOM tool adds transformation primitives to Java, such that libraries for analysis can be used directly. In either approach, the job of integrating analysis with transformation has to be done over and over again for each application and this requires a significant investment.

We propose a more radical solution by completely merging the set of concepts for analysis and transformation of source code into a single language called Rascal. This language covers the range of applications from pure analyses to pure transformations and everything in between. Our contribution does not consist of new concepts or language features *per se*, but rather the careful collaboration, integration and cross-fertilization of existing concepts and language features.

3.4.1. Goals

The goals of Rascal are: (a) to remove the cognitive and computational overhead of integrating analysis and transformation tools, (b) to provide a safe and interactive environment for constructing and experimenting with large and complicated source code analyses and transformations such as, for instance, needed for refactorings, and (c) to be easily understandable by a large group of computer programming experts. Rascal is not limited to one particular object programming language, but is generically applicable. Reusable, language specific, functionality is realized as libraries. As an end-result we envision Rascal to be a one-stop shop for source code analysis, transformation, generation and visualization.

3.5. Domain-specific Languages

Our final goal is centered around Domain-specific languages (DSLs), which are software languages tailored to a specific problem domain. DSLs can provide orders of magnitude improvement in terms of software quality and productivity. However, the implementation of DSLs is challenging and requires not only thorough knowledge of the problem domain (e.g., finance, digital forensics, insurance, auditing etc.), but also knowledge of language implementation (e.g., parsing, compilation, type checking etc.). Tools for language implementation have been around since the archetypical parser generator YACC. However, many of such tools are characterized by high learning curves, lack of integration of language implementation facets, and lead to implementations that are hard to maintain. This line of research focuses on two topics: improve the practice and experience of DSL implementation, and evaluate the success of DSLs in industrial practice.

Language workbenches [5] are integrated environments to facilitate the development of all aspects of DSLs. This includes IDE support (e.g., syntax coloring, outlining, reference resolving etc.) for the defined languages. Rascal can be seen as a language workbench that focuses on flexibility, programmability and modularity. DSL implementation is, in essence, an instance of source code analysis and transformation. As a result, Rascal's features for fact extraction, analysis, tree traversal and synthesis are an excellent fit for this area. An important aspect in this line of research is bringing the IDE closer to the source code. This will involve investigation of heterogeneous representations of source code, by integrating graphical, tabular or forms-based user interface elements. As a result, we propose Rascal as a feature-rich workbench for model-driven software development.

The second component of this research is concerned with evaluating DSLs in industrial contexts. This means that DSLs constructed using Rascal will be applied in real-life environments so that expected improvements in quality, performance, or productivity can be observed. We already have experience with this in the domain of digital forensics, computational auditing and games.

3.5.1. Goals

The goal of this research topic is to improve the practice of DSL-based software development through language design and tool support. A primary focus is to extend the IDE support provided by Rascal, and to facilitate incremental, and iterative design of DSLs. The latter is supported by new (meta-)language constructs for extending existing language implementations. This will require research into extensible programming and composition of compilers, interpreters and type checkers. Finally, a DSL is never an island: it will have to integrate with (third-party) source code, such as host language, libraries, runtime systems etc. This leads to the vision of multi-lingual programming environments [15].

4. Highlights of the Year

4.1. Highlights of the Year

4.1.1. Awards

Prof.dr. Paul Klint won the IEEE TCSE Software Engineering Distinguished Service Award 2015. This award is presented "annually for outstanding and/or sustained contributions and service to the software engineering community".

5. New Software and Platforms

5.1. MicroMachinations

FUNCTIONAL DESCRIPTION

Objective: To create an integrated, live environment for modelling and evolving game economies. This will allow game designers to experiment with different strategies to realise game mechanics. The environment integrates with the SPIN model checker to prove properties (reachability, liveness). A runtime system for executing game economies allows MicroMachinations models to be embedded in actual games.

Impact: One of the important problems in game software development is the distance between game design and implementation in software. MicroMachinations has the potential to bridge this gap by providing live design tools that directly modify or create the desired software behaviours.

- Participants: Paul Klint and Riemer Van Rozen
- Contact: Riemer Van Rozen
- URL: https://github.com/vrozen/MM-Lib

5.2. OSSMETER

KEYWORDS: Software Quality, Metrics, Open-source SCIENTIFIC DESCRIPTION: OSSMETER meets the challenge of software project quality assessment via fact-based business intelligence. The goal of the project was to design and evaluate a platform for incremental analysis of long lasting open-source projects to support decision making on the corporate level. FUNCTIONAL DESCRIPTION: OSSMETER is a platform which integrates metrics of open-source projects: their source code quality, the contents of their social interactions and their activity in issue tracking systems. It includes a fully programmable user-defined quality model utility and configurable dash-board user-interface. The basic metrics of the platform and their aggregation to the project level are carefully considered and rationalised.

- Participants: Paul Klint, Jurgen Vinju, Tijs Van Der Storm, Ashim Shahi, Bas Basten.
- Contact: Jurgen Vinju
- URL: http://www.ossmeter.org/

5.3. Rascal

KEYWORDS: Metaprogramming - Language
SCIENTIFIC DESCRIPTION

Rascal primitives include immutable data, context-free grammars and algebraic data-types, relations, relational calculus operators, advanced patterns matching, generic type-safe traversal, comprehensions, concrete syntax for objects, lexically scoped backtracking, and string templates for code generation. It has libraries for integrating language front-ends, for reusing analysis algorithms, for getting typed meta-data out of version management systems, for interactive visualization, etc.
FUNCTIONAL DESCRIPTION

Rascal is a programming language, such that meta programs can be created by, understood by, and debugged by programmers.

You want to use the best tool for the job when analyzing, transforming or generating source code, so normally you will end up with many different tools, possibly even written in different languages. Now the problem is to integrate these tools again. Rascal solves this problem by integrating source code analysis, transformation, and generation primitives on the language level. Use it for any kind of metaprogramming task: to construct parsers for programming languages, to analyze and transform source code, or to define new DSLs with full IDE support.

- Participants: Paul Klint, Jurgen Vinju, Tijs Van Der Storm, Davy Landman, Bert Lisser, Atze Van Der Ploeg, Vadim Zaytsev, Anastasia Izmaylova, Michael Steindorfer, Jouke Stoel, Ali Afroozeh and Ashim Shahi
- Contact: Paul Klint
- URL: http://www.rascal-mpl.org/

5.4. Meerkat

FUNCTIONAL DESCRIPTION

Objective: To enable fully context-free general parsing using a parser combinator library (including allowing left recursion and arbitrary context-sensitive disambiguation).

Impact: Meerkat explores algorithmic advances in context-free general parsing (based on the GLL parsing algorithm and memoized continuations) in the context of a scala parsing combinator library. This library uniquely combines the worst-case execution time guarantees of GLL with the flexibility of parsing combinators. [47]

- Participants: Anastasia Izmaylova, Ali Afroozeh and Tijs van der Storm.
- Contact: Anastasia Izmaylova, Ali Afroozeh
- URL: http://meerkat-parser.github.io/

5.5. Iguana

FUNCTIONAL DESCRIPTION

Objective: To provide a data-dependent context-free general parsing infra-structure for parsing programming languages and other formal data, program and modeling notations.

Impact: Iguana is a fast implementation of data-dependent grammars based on the GLL context-free parsing algorithm with data-dependent non-terminals and constraints on top. It comes with a number of high-level disambiguation constructs which are translated to the intermediate layer of data-dependent (E)BNF before being loaded into an object-oriented implementation of GLL based on abstract transition network. Using Iguana parsers for languages which are considered to be hard to parse (such as Haskell and OCAML) are within reach of being generated from simple declarative specifications [25].

- Participants: Anastasia Izmaylova, Ali Afroozeh.
- Contact: Anastasia Izmaylova, Ali Afroozeh
- URL: http://iguana-parser.github.io/

5.6. Capsule

FUNCTIONAL DESCRIPTION

Objective: A generic and highly optimised product-family of immutable collection data-structures.

Impact: Capsule is a library for immutable sets, maps and tables. The code is generated using high-level descriptions of the requirements and internal trade-offs of hash-trie map based implementations. We are using this code generator to experiment with the fastest and leanest representations of these persistent data-types to satisfy the requirements of Rascal meta-programming applications in static analysis, empirical research in software engineering and software analytics [37].

- Participants: Michael Steindorfer, Jurgen Vinju
- Contact: Michael Steindorfer
- URL: http://usethesource.io/projects/capsule/

6. New Results

6.1. Faster Immutable Data Structures for the JVM

Immutable data structures involve copying when updating. Efficient implementations use persistent data-structures, so that most of the unchanged data is shared between the copies. Existing libraries for such data structures in the context of the Java virtual machine (JVM), such as the data structures in Clojure and Scala, are based on Hash Array-Mapped Tries (HAMTs), which provide efficient insertion and concatenation operations for persistent maps and sets. In [37] Steindorfer and Vinju presented additional optimisation which allow such operations to be up to 28 times faster than in the Clojure and Scala libraries. Furthermore, the cost of equality checking of such data structures is lower as well. All this, without incurring additional memory.

6.2. Automated Measurement and Analysis of Open Source Software

Deciding whether an open source software (OSS) meets the required standards for adoption in terms of quality, maturity, activity of development and user support is not a straightforward process. It involves analysing various sources of information, including the project's source code repositories, communication channels, and bug tracking systems. OSSMETER extends state-of-the-art techniques in the field of automated analysis and measurement of open-source software (OSS), and develops a platform that supports decision makers in the process of discovering, comparing, assessing and monitoring the health, quality, impact and activity of opensource software. To achieve this, OSSMETER computes trustworthy quality indicators by performing advanced analysis and integration of information from diverse sources including the project metadata, source code repositories, communication channels and bug tracking systems of OSS projects [29], [26]

This result comes from intensive collaboration in the FP7 STREP project "OSSMETER". The ATEAMS contribution is focused around source code metrics and activity analysis for Java and PHP.

6.3. Modular Interpreters for the Masses

Object Algebras [46] are new design pattern for increased modularity and extensibility of tree based, abstract data types. By modelling the abstract syntax of a language as a generic factory interface, implementations of this interface provide multiple semantics of the data. For instance, one can define evaluation, type checking and pretty printing of the abstract syntax fully modularly. Additionally, the pattern allows syntax extension: adding a new constructor to the datatype, and modularly extending any existing interpretations to deal with the construct. The same interpretation of different constructs, however, might involve different kinds of context information. For instance, evaluation of arithmetic expressions does not require any context information, but

evaluation of variables and binders requires and environment. In [34], Inostroza and Van der Storm introduce a simple, modular, and type safe technique to allow such interpretations to be composed anyway. It is based on lifting one interpreter to implicitly propagate the context information it does not require, so that the signatures of the interpreters become compatible. As a result, semantic definitions of language modules do not have to anticipate all kinds of context information that might be required by other modules with which it might be composed. The technique is simple, does not sacrifice separate compilation, is easy automate, and works in mainstream languages. It provides a first step towards a foundation for defining language by assembling modular building blocks.

6.4. One Parser to Rule Them All

Parsing realistic languages requires much more than just a parsing algorithm. Different kinds of language require advanced disambiguation, operator priorities, off-side rule checking, whitespace dependence or data dependence. In [25], Afroozeh and Izmaylova showed how most of these concerns are actually instances of data dependent parsing: the parsing process depends on the value of previously parsed input. They provided an encoding of indentation sensitive parsing, operator precedence and parsing in the presence of preprocessor directives, to a simple, data dependent core language which is executed using the general parsing algorithm GLL. By exposing the data dependent machinery at the level of the grammar formalism , this opens up a range of possibilities for custom parsing aspects, and provides a clear semantics for existing concerns like disambiguation.

6.5. A Pattern-Based Game Mechanics Design Assistant

Video game designers iteratively improve player experience by play testing game software and adjusting its design. Deciding how to improve gameplay, however, is difficult and time-consuming: designers lack an effective means for exploring decision alternatives and modifying a game's mechanics. In [35], Van Rozen presented the Mechanics Pattern Language (MPL) for encoding common game economy structures and design intent, and a Mechanics Design Assistant (MeDeA) for analyzing, explaining, understanding existing mechanics, and generating, filtering, exploring and applying design alternatives for modifying mechanics. As a result, game designers' productivity and game quality is increased by providing feedback and design alternatives early in the development cycle. Furthermore, the game economy modifications are applied at runtime using the MicroMachinations library, so that the effect of changes can be immediately experienced.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Grants with Industry

With the **ING bank** we a running a four-year project on advising and research in functional and non-functional properties of a part of the ING IT-infrastructure. The project involves modelling a large part of the product portfolio and using state-of-the-art MDE technology to simulate, verify and generate part of its IT infrastructure. The funding of this project is approximately 50% industry, 50% grants from CWI & NWO.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. Master Software Engineering

ATEAMS is a core partner in the Master Software Engineering at Universiteit van Amsterdam. This master is a collaboration between SWAT/ATEAMS, Universiteit van Amsterdam, Vrije Universiteit and Hogeschool van Amsterdam.

8.1.2. Early Quality Assurance in Software Production

The EQUA project is a collaboration among Hogeschool van Amsterdam (main partner) Centrum Wiskunde & Informatica (CWI), Technisch Universiteit Delft, Laboratory for Quality of Software (LaQuSo), Info Support, Software Improvement Group (SIG), and Fontys Hogeschool Eindhoven.

8.1.3. Next Generation Auditing: Data-assurance as a service

This project is a collaboration between Centrum Wiskunde & Informatica (CWI) PriceWaterhouseCoopers (PWC), Belastingdienst (National Tax Office), and Computational Auditing, is to enable research in the field of computational auditing.

8.1.4. Domain-Specific Languages: A Big Future for Small Programs

Software and programming have a brilliant past that has brought us the automation of many expected and unexpected human and societal activities ranging from banking and consumer electronics to mobile networking, search engines and social networks. The present of software is overwhelming: many software systems have sizes in the range of 10–100 million lines of source code and contain tens of thousands of errors that are yet to be discovered. We claim that software will only have a big future if software itself becomes smaller. Smaller software leads to higher software productivity (we have to write less) and higher software quality (quality guarantees become part of the language and not of the program).

This project is funded by NWO (the Dutch national science foundation).

8.2. European Initiatives

8.2.1. FP7 & H2020 Projects

- FP7 STREP “OSSMETER — Automated Measurement and Analysis of Open Source Software” (ended in 2015)

8.2.2. Collaborations with Major European Organizations

Centrum Wiskunde & Informatica (CWI): Software Analysis & Transformation (Netherlands)

CWI SWAT is the research team associated directly with ATEAMS.

8.3. International Initiatives

8.3.1. Inria International Partners

8.3.1.1. Informal International Partners

ATEAMS collaborates with the following research teams:

- Eindhoven Technical University - SET (Eindhoven, The Netherlands)
- Universiteit van Amsterdam - Systems and Network Engineering (Amsterdam, The Netherlands)
- Royal Holloway University of London - Dept. of Computer Science
- The University of Hong Kong (China) - Computer Science
- Delft Technical University (The Netherlands)
- University of Texas at Austin (USA)
- TU Darmstadt (Germany)

8.3.1.2. Research stays abroad

- Michael Steindorfer stayed for 3 months at Oracle Labs in Austria to study efficient data-structures and data-structure optimisations on the JVM.

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific events organisation

9.1.1.1. General chair, scientific chair

- Tijs van der Storm: co-organizer of the 3rd Workshop on Domain-Specific Language Design and Implementation (DSLDI'15).

9.1.1.2. Member of the organizing committees

- Tijs van der Storm: Publicity co-chair SPLASH and track organizer SPLASH-I
- Jurgen Vinju: co-organizer Bits & Chips Software Event (Legacy Software Track), Eindhoven, The Netherlands

9.1.2. Scientific events selection

9.1.2.1. Member of the conference program committees

- Paul Klint: International Conference on Software Language Engineering (SLE'15), 14th edition of the BElgian-NEtherlands software eVOLution seminar (BENEVOL 2015), 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)
- Tijs van der Storm: International Conference on Model Transformation (ICMT'15), International Conference on Software Language Engineering (SLE'15), Transformation Tool Context (TTC'15), Future Programming Workshop (FPW'15).
- Jurgen Vinju: 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15), 3rd International Workshop on The Globalization of Modeling Languages Workshop (GEMOC'15), 5th Summer School on Grand Timely Topics in Software Engineering (GTTSE'15), IEEE International Conference on Software Comprehension (ICPC'15), 14th edition of the BElgian-NEtherlands software eVOLution seminar (BENEVOL 2015).

9.1.2.2. Reviewer

9.1.2.2.1. Journal

- Member of the editorial boards:
  - Paul Klint: editor of Science of Computer Programming (Elsevier SCP)
- Reviewer - Reviewing activities:
  - Paul Klint: Science of Computer Programming
  - Tijs van der Storm: Science of Computer Programming (SCP), Computer Languages and Systems (COMLAN)
  - Jurgen Vinju: Science of Computer Programming (SCP), Computer Languages and Systems (COMLAN), Journal on Empirical Software Engineering (ESE), Journal of Software Maintenance and Evolution (JSME)

9.1.3. Invited talks

- Textual and projectional language workbenches. Tijs van der Storm, Dagstuhl on "Domain-Specific Languages".

- "Software Engineering: The War Against Complexity", Jurgen Vinju. CHA-Q Open Tool Demonstrations Event (keynote), February 24, Antwerpen, Belgium.
- "Program Analysis and Transformation with Rascal" tutorial at the 36th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI) in Portland, USA. Mark Hills, Jurgen Vinju, Paul Klint.

9.1.4. Leadership within the scientific community

9.1.4.1. Member of steering committees

- Jurgen Vinju: International Conference on Software Language Engineering (ACM SLE) (*chair*)
- Jurgen Vinju: International Working Conference on Source Analysis and Manipulation (IEEE SCAM)

9.1.4.2. Member of other groups

- Jurgen Vinju: Member of IFIP WG 2.3 — Working Group on Software Implementation Technology
- Tijs van der Storm: Member of IFIP WG 2.16 — Working Group on Language Design
- Tijs van der Storm: Member of the board of European Association for Programming Languages and Systems (EAPLS)

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Tijs van der Storm, "Software Construction", Masters, Universiteit van Amsterdam

Jurgen J. Vinju, "Software Evolution", Masters, Universiteit van Amsterdam

9.2.2. Supervision

PhD : Atze van der Ploeg, " Efficient Abstractions for Visualization and Interaction", Universiteit van Amsterdam, April, 8th, 2015 [48] (advisors: Tijs van der Storm and Paul Klint)

PhD in progress : Ali Afroozeh, "Closing the Book on Parsing", 2017 (advisors: Jurgen Vinju and Paul Klint)

PhD in progress : Pablo Inostroza Valdera, "Modular Language Implementation with Object Algebras", 2018 (advisors: Tijs van der Storm and Paul Klint)

PhD in progress : Anastasia Izmaylova, "Parser Combinators Revisited", 2017 (advisors: Jurgen Vinju and Paul Klint)

PhD in progress : Davy Landman, "Source Code Complexity in Context" , 2016 (advisros: Jurgen Vinju and Paul Klint)

PhD in progress : Riemer van Rozen, "Domain-Specific Languages for Game Developmenr", 2017 (advisors: Paul Klint and Tijs van der Storm)

PhD in progress : Michael Steindorfer, "Fast and Lean Immutable Data Structures", 2017 (advisors: Jurgen Vinju and Paul Klint)

PhD in progress : Jouke Stoel, "Solving the Bank", 2019 (advisors: Jurgen Vinju and Tijs van der Storm)

Msc : Zisimopolous, P.: "A Grid Scheduling infrastructure for SmartConnect's performance monitoring calculations", Universiteit van Amsterdam, The Netherlands

Msc : Zhelyazkov, A.T.: "Form controls in WebGL. A stepping stone to a WebGL library for developing commercial interactive 3D websites", Universiteit van Amsterdam, The Netherlands

Msc : Valencia Vargas, S.: "Begelman vs. FolkRank. The Comparison of Two Algorithms in the Tag Recommendation Context: An Exploratory Study", Universiteit van Amsterdam, The Netherlands

Msc : Heimensen, M.: “JavaScript language extension with language workbenches”, Universiteit van Amsterdam, The Netherlands

Msc : Harezlak, H.: “Geographically-aware scaling for real-time persistent WebSocket applications”, Universiteit van Amsterdam, The Netherlands

Msc: Chow, K.: “Performance of Face Recognition Algorithms on Mobile Devices”, Universiteit van Amsterdam, The Netherlands

Msc: Iwan Flameling: “An automatic CSRF protection tool”, Universiteit van Amsterdam, The Netherlands

Msc: Omar Pakker: “Graph-Based Querying On top of the Entity Framework”, Universiteit van Amsterdam, The Netherlands

9.2.3. Juries

- Jurgen Vinju:
  - Phd Reza Yazdanshenas, A. — Universitet i Oslo, Norway
  - Phd Hafeez Osman, M. — Universiteit Leiden, The Netherlands

9.3. Popularization

- Ali Afroozeh and Anastasia Izmaylova: “Meerkat parsers: a general parser combinator library for real programming languages”, Scala Days 2015, Amsterdam, The Netherlands.
- Paul Klint: “Paul Klint in RTL Nieuws over het nieuwe betalingssysteem van het pgb” (radio interview).
- Paul Klint: “De evolutie van codetaal”, media appearance, NRC Handelsblad
- Paul Klint: “Première internetfilm over Nederlands informatica erfgoed”, media appearance, Automatisering Gids.
- Paul Klint: “Remembering Arra: a pioneer in Dutch computing”, media appearance, http://www.engineersonline.nl/.
- Davy Landman: “Let’s talk about complexity”, Devnology software development community talk, Amsterdam, The Netherlands.
- Tijs van der Storm: “Opportunities and Risks of MDSE: experience with Derric, a DSL for Digital Forensics”, Bits&Chips industry conference.
- Jouke Stoel and Tijs van der Storm: “Hack Your Language with Rascal”, workshop at Joy of Coding conference, Rotterdam, The Netherlands.
- Tijs van der Storm, “Live Little Languages”, JBI Colloquium, Groningen, The Netherlands.
- Jurgen Vinju: “Public/Private Collaboration {in,for,with} Software Engineering”, 21st Annual Conference of the European Association of Research Managers and Administrators , Leiden, The Netherlands
- Jurgen Vinju: OSSMETER Pitch EU Concertation Meeting - Turning cloud research into innovative software & services, March 25th 2015, Brussels, Belgium.
- Jurgen Vinju: Challenges and Opportunities of Big Software-based Innovation NWO Big Software Match Making Day, July 1st, 2015, Utrecht, The Netherlands.

10. Bibliography

Major publications by the team in recent years

[1] B. BASTEN. *Tracking Down the Origins of Ambiguity in Context-Free Grammars*, in "Seventh International Colloquium on Theoretical Aspects of Computing (ICTAC 2010)", A. CAVALCANTI, D. DEHARBE, M.-C. GAUDEL, J. WOODCOCK (editors), Springer, September 2010, vol. 6255, pp. 76-90

[2] P. CHARLES, R. M. FUHRER, S. M. SUTTON JR, E. DUESTERWALD, J. VINJU. *Accelerating the Creation of Customized, Language-Specific IDEs in Eclipse*, in "Proceedings of the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009", S. ARORA, G. T. LEAVENS (editors), 2009

[3] B. C. DOS SANTOS OLIVEIRA, T. VAN DER STORM, A. LOH, W. R. COOK. *Feature-Oriented Programming With Object Algebras*, in "Proceedings of the European Conference on Object-Oriented Programming (ECOOP)", 2013, http://hal.inria.fr/hal-00923387

[4] J. V. EIJCK, C. UNGER. *Computational Semantics with Functional Programming*, Cambridge University Press, September 2010

[5] S. ERDWEG, T. V. D. STORM, M. VOELTER, L. TRATT, R. BOSMAN, W. R. COOK, A. GERRITSEN, A. HULSHOUT, S. KELLY, A. LOH, G. KONAT, P. J. MOLINA, M. PALATNIK, R. POHJONEN, E. SCHINDLER, K. SCHINDLER, R. SOLMI, V. VERGU, E. VISSER, K. B. V. D. VLIST, G. WACHSMUTH, J. M. V. D. WONING. *Evaluating And Comparing Language Workbenches: Existing Results And Benchmarks For The Future*, in "Computer Languages, Systems and Structures", 2015, vol. 44, n^{o} Part A, pp. 24 - 47, https://hal.inria.fr/hal-01261481

[6] M. HILLS, P. KLINT, J. VINJU. *Program Analysis Scenarios In Rascal*, in "Proceedings of the International Workshop on Rewriting Logic and its Applications (WRLA, 2012)", Talinn, Estonia, F. DURÁN (editor), Springer, 2012, vol. 7571, pp. 10 - 30, An invited paper for WRLA 2012, describing our work on program analysis and comparing our approach to approaches based on rewriting logic semantics, http://hal.inria.fr/hal-00756880

[7] M. HILLS, P. KLINT, J. VINJU. *Scripting A Refactoring With Rascal And Eclipse*, in "Proceedings of the 5th Workshop on Refactoring Tools 2012", Rapperswil, Switzerland, P. SOMMERLAD (editor), ACM, 2012, pp. 40 - 49, http://hal.inria.fr/hal-00756879

[8] M. HILLS, P. KLINT, T. VAN DER STORM, J. VINJU. *A One-Stop Shop For Software Evolution Tool Construction*, in "ERCIM News", 2012, n^{o} 88, pp. 11 - 12, http://hal.inria.fr/hal-00756876

[9] A. IZMAYLOVA, P. KLINT, A. SHAHI, J. VINJU. *M3: An Open Model For Measuring Code Artifacts*, 2013, n^{o} arXiv-1312.1188, pp. 1-2, https://hal.inria.fr/hal-00923379

[10] P. KLINT, T. V. D. STORM, J. VINJU. *RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation*, in "IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'09)", Los Alamitos, CA, USA, 2009, pp. 168-177, http://doi.ieeecomputersociety.org/10.1109/SCAM.2009.28

[11] P. KLINT, T. VAN DER STORM, J. VINJU. *EASY Meta-programming with Rascal*, in "Generative and Transformational Techniques in Software Engineering III", J. FERNANDES, R. LÄMMEL, J. VISSER, J. SARAIVA (editors), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, vol. 6491, pp. 222-289, http://dx.doi.org/10.1007/978-3-642-18023-1_6

[12] P. KLINT, R. VAN ROZEN. *Micro-Machinations: A DSL For Game Economies*, in "Proceedings of the International Conference on Software Language Engineering (SLE, 2013)", Unknown, M. ERWIG, R. F. PAIGE, E. VAN WYK (editors), Lecture Notes in Computer Science, Springer, 2013, vol. 8225, pp. 36 - 55, https://hal.inria.fr/hal-00923383

[13] A. LOH, T. VAN DER STORM, W. R. COOK. *Managed Data: Modular Strategies For Data Abstraction*, in "Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software 2012", Tucson, United States, ACM, 2012, pp. 179 - 194, http://hal.inria.fr/hal-00756886

[14] M. J. STEINDORFER, J. J. VINJU. *Optimizing Hash-array Mapped Tries for Fast and Lean Immutable JVM Collections*, in "Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications", New York, NY, USA, OOPSLA 2015, ACM, 2015, pp. 783–800, http://doi.acm.org/10.1145/2814270.2814312

[15] T. VAN DER STORM, J. VINJU. *Towards Multilingual Programming Environments*, in "Science of Computer Programming", 2013, https://hal.inria.fr/hal-00923385

[16] T. VAN DER STORM, W. R. COOK, A. LOH. *Object Grammars: Compositional & Bidirectional Mapping Between Text and Graphs*, in "Software Language Engineering", Dresden, Germany, K. CZARNECKI, G. HEDIN (editors), September 2012, http://hal.inria.fr/hal-00758627

[17] J. VINJU, M. W. GODFREY. *What does control flow really look like? Eyeballing the Cyclomatic Complexity Metric*, in "Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'12)", IEEE Computer Society, 2012

[18] J. VAN DEN BOS, T. VAN DER STORM. *Bringing Domain-Specific Languages to Digital Forensics*, in "Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011", Honolulu, United States, ACM, 2011, pp. 671-680, http://hal.inria.fr/hal-00644687/en

[19] J. VAN DEN BOS, T. VAN DER STORM. *Domain-Specific Languages For Better Forensic Software*, in "ERCIM News", 2012, vol. 2012, n° 90, http://hal.inria.fr/hal-00756885

[20] J. VAN DEN BOS, T. VAN DER STORM. *Domain-Specific Optimization In Digital Forensics*, in "Proceedings of the International Conference on Model Transformation (ICMT, 2012)", Prague, Czech Republic, Z. HU, J. DE LARA (editors), Springer, 2012, vol. 7307, pp. 121 - 136, http://hal.inria.fr/hal-00756891

Publications of the year

Articles in International Peer-Reviewed Journals

[21] H. J. S. BASTEN, J. V. D. BOS, M. A. HILLS, P. KLINT, A. W. LANKAMP, B. LISSER, A. J. V. D. PLOEG, T. V. D. STORM, J. J. VINJU. *Modular Language Implementation In Rascal — Experience Report*, in "Science of Computer Programming", December 2015, vol. 114, pp. 7 - 19, https://hal.inria.fr/hal-01261480

[22] J. V. EIJCK, T. V. D. STORM. *Understanding Information Update In Questionnaires*, in "Science of Computer Programming", January 2015, vol. 97, n° Part 1, https://hal.inria.fr/hal-01261475

[23] S. ERDWEG, T. V. D. STORM, M. VOELTER, L. TRATT, R. BOSMAN, W. R. COOK, A. GERRITSEN, A. HULSHOUT, S. KELLY, A. LOH, G. KONAT, P. J. MOLINA, M. PALATNIK, R. POHJONEN, E. SCHINDLER, K. SCHINDLER, R. SOLMI, V. VERGU, E. VISSER, K. B. V. D. VLIST, G. WACHSMUTH, J. M. V. D. WONING. *Evaluating And Comparing Language Workbenches: Existing Results And Benchmarks For The Future*, in "Computer Languages, Systems and Structures", 2015, vol. 44, n° Part A, pp. 24 - 47, https://hal.inria.fr/hal-01261481

[24] T. V. D. STORM, J. J. VINJU. *Towards Multilingual Programming Environments*, in "Science of Computer Programming", January 2015, vol. 97, n^{o} Part 1, https://hal.inria.fr/hal-01261474

International Conferences with Proceedings

[25] A. AFROOZEH, A. IZMAYLOVA. *One Parser to Rule Them All*, in "Proceedings of the 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software", New York, NY, USA, Onward! 2015, ACM, 2015, pp. 151–170 [*DOI :* 10.1145/2814228.2814242], https://hal.inria.fr/hal-01261484

[26] B. ALMEIDA, S. ANANIADOU, A. BAGNATO, A. BERRETEAGA, J. DI ROCCO, D. DI RUSCIO, D. KOLOVOS, I. KORKONTZELOS, S. HANSEN, P. MALÓ, N. MATRAGKAS, R. F. PAIGE, J. J. VINJU. *OSSMETER: Automated Measurement And Analysis Of Open Source Software*, in "Proceedings of International Conference on Software Technologies: Applications and Foundations 2015 (STAF 0)", L'Aquila, Italy, Lecture Notes in Computational Science and Engineering, Springer, 2015, https://hal.archives-ouvertes.fr/hal-01261966

[27] H. J. S. BASTEN, M. A. HILLS, P. KLINT, D. LANDMAN, A. SHAHI, M. STEINDORFER, J. J. VINJU. *M3: A General Model For Source Code Analytics In Rascal*, in "Proceedings of International Workshop on Software Analytics 2015 (SWAN 2015)", Montreal, Canada, IEEE, March 2015, https://hal.inria.fr/hal-01261493

[28] J. V. BENTHEM, J. V. EIJCK, M. GATTINGER, K. SU. *Symbolic Model Checking for Dynamic Epistemic Logic*, in "Logic, Rationality, and Interaction; 5th International Workshop, LORI 2015", Taipei, Taiwan, W. V. D. HOEK, H. H. WESLEY, W. WEN-FANG (editors), LNCS, Springer, 2015, n^{o} 9394, pp. 366–378, https://hal.inria.fr/hal-01261492

[29] D. DIRUSCIO, D. KOLOVOS, N. MATRAGKAS, I. KORKONTZELOS, J. J. VINJU. *OSSMETER: A Software Measurement Platform For Automatically Analysing Open Source Software Projects*, in "Proceedings of Joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering 2015 (ESEC/FSE)", Bergame, Italy, ACM International Conference Proceeding Series, ACM, 2015, https://hal.archives-ouvertes.fr/hal-01261967

[30] J. V. EIJCK, M. GATTINGER. *Elements of Epistemic Crypto Logic (Extended Abstract)*, in "Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)", Istanbul, Turkey, BORDINI, ELKIND, WEISS, YOLUM (editors), 2015, https://hal.inria.fr/hal-01261489

[31] M. GATTINGER, J. V. EIJCK. *Towards Model Checking Cryptographic Protocols with Dynamic Epistemic Logic*, in "Proceedings LAMAS (LAMAS 2015)", Istanbul, Turkey, 2015, available from http://www.irit.fr/~Emiliano.Lorini/LAMAS2015/accepted.htm, https://hal.inria.fr/hal-01261491

[32] C. HENTZ, J. J. VINJU, A. M. MOREIRA. *Reducing The Cost Of Grammar-Based Testing Using Pattern Coverage*, in "Proceedings of IFIP International Conference on Testing Software and Systems 2015 (ICTSS)", Dubai, United Arab Emirates, Springer, 2015, https://hal.archives-ouvertes.fr/hal-01261968

[33] F. HERMANS, T. V. D. STORM. *Copy-Paste Tracking: Fixing Spreadsheets Without Breaking Them*, in "Proceedings of the International Conference on Live Coding (ICLC, 2015)", Leeds, UK, 2015, https://hal.inria.fr/hal-01261473

[34] P. A. INOSTROZA VALDERA, T. V. D. STORM. *Modular Interpreters For The Masses: Implicit Context Propagation Using Object Algebras*, in "Proceedings of ACM International Conference on Generative Programming and Component Engineering 2015 (GPCE 0)", C. KÄSTNER, A. GOKHĀLÉ (editors), ACM International Conference Proceeding Series, ACM, 2015, pp. 171 - 180, https://hal.inria.fr/hal-01261476

[35] R. V. ROZEN. *A Pattern-Based Game Mechanics Design Assistant*, in "Proceedings of Foundations of Digital Games 2015 (FDG 2015)", Pacific Grove, United States, Society for the Advancement of the Science of Digital Games, 2015, at Asilomar Conference Grounds, https://hal.archives-ouvertes.fr/hal-01261970

[36] R. V. ROZEN, T. V. D. STORM. *Origin Tracking + Text Differencing = Textual Model Differencing*, in "Theory and Practice of Model Transformations", Springer International Publishing, 2015, pp. 18 - 33, https://hal.inria.fr/hal-01261479

[37] M. J. STEINDORFER, J. J. VINJU. *Optimizing Hash-array Mapped Tries for Fast and Lean Immutable JVM Collections*, in "Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications", New York, NY, USA, OOPSLA 2015, ACM, 2015, pp. 783–800 [*DOI :* 10.1145/2814270.2814312], https://hal.inria.fr/hal-01261487

[38] J. H. STOEL. *A Case For Rebel, A DSL For Product Specifications*, in "Proceedings of Domain-specific Language Design and Implementation 2015 (DSLDI 0)", arXiv, 2015, pp. 9 - 11, https://hal.archives-ouvertes.fr/hal-01261969

[39] H. ZHANG, Z. CHU, B. C. DOS SANTOS OLIVEIRA, T. V. D. STORM. *Scrap Your Boilerplate With Object Algebras*, in "Proceedings of the Object-oriented Programming, Systems, Languages, and Applications (OOPSLA, 2015)", 2015, https://hal.inria.fr/hal-01261477

Scientific Books (or Scientific Book chapters)

[40] T. V. D. STORM, S. ERDWEG (editors). *Proceedings Of The 3rd Workshop On Domain-Specific Language Design And Implementation (DSLDI'15)*, August 2015, https://hal.inria.fr/hal-01261478

[41] A. AFROOZEH, A. IZMAYLOVA. *Faster, Practical GLL Parsing*, in "Compiler Construction", B. FRANKE (editor), Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2015, vol. 9031, pp. 89-108 [*DOI :* 10.1007/978-3-662-46663-6_5], https://hal.inria.fr/hal-01261483

[42] J. V. BENTHEM, H. V. DITMARSCH, J. V. EIJCK, J. JASPARS. *Logic in Action*, Internet, 2015, electronic book, available from url below, https://hal.inria.fr/hal-01261490

[43] J. V. EIJCK. *Implementing Semantic Theories*, in "Handbook of Contemporary Semantics, Second Edition", S. LAPPIN, C. FOX (editors), Wiley, 2015, pp. 455–491, https://hal.inria.fr/hal-01261485

[44] J. V. EIJCK. *Strategies in Social Software*, in "Modeling Strategic Reasoning: Logics, Games and Communities", J. VAN BENTHEM, S. GHOSH, R. VERBRUGGE (editors), LNCS, Springer, 2015, n^{o} 8972, https://hal.inria.fr/hal-01261486

[45] J. V. EIJCK. *Varieties of Belief and Probability*, in "The Facts Matter —- Essays on Logic and Cognition in Honour of Rineke Verbrugge", S. GHOSH, J. SZYMANIK (editors), Tributes, Volume 25, College Publications, 2015, pp. 67–87, https://hal.inria.fr/hal-01261488

References in notes

[46] B. C. DOS SANTOS OLIVEIRA, W. R. COOK. *Extensibility for the Masses*, in "ECOOP 2012–Object-Oriented Programming", Springer, 2012, pp. 2–27

[47] A. IZMAYLOVA, A. AFROOZEH, T. V. D. STORM. *Practical, General Parser Combinators*, in "Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation", New York, NY, USA, PEPM 2016, ACM, 2016, pp. 1–12, http://doi.acm.org/10.1145/2847538.2847539

[48] A. J. V. D. PLOEG. *Efficient Abstractions For Visualization And Interaction*, Universiteit van Amsterdam, April 2015, pp. 1 - 146, http://oai.cwi.nl/oai/asset/23618/23618A.pdf