INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team ATEAMS

Analysis and Transformation based on rEliAble tool coMpositionS

Lille - Nord Europe

Theme : Programs, Verification and Proofs

Table of contents

This is the report for ATEAMS of INRIA Lille. ATEAMS is located in Amsterdam and is hosted by Centrum Wiskunde & Informatica (CWI). The group is funded equally —as a principle— both by CWI and INRIA. We are proud to represent this very close collaboration between our two excellent European research institutes.

1. Team

Research Scientists

Paul Klint [Group leader SEN1, Head of Software Engineering Departement CWI, Director Master Software Engineering at Universiteit van Amsterdam, Team Leader, HdR]
Jan van Eijck [Senior Researcher CWI, Professor Universiteit Utrecht, HdR]
Jurgen Vinju [Senior Researcher CWI, Teacher Universiteit van Amsterdam, HdR]
Tijs van der Storm [Senior Researcher CWI, Teacher Universiteit van Amsterdam]

Technical Staff

Bert Lisser [Scientific Programmer CWI, HdR]
Arnold Lankamp [Scientific Programmer CWI]
Maarten Dijkema [Technical support CWI, HdR]

PhD Students

Bas Basten [Junior Researcher CWI]
Jeroen van den Bos [Junior Researcher CWI]
Yanying Wang [Junior Researcher CWI]
Floor Sietsma [Junior Researcher CWI]

Post-Doctoral Fellow

Mark Hills [Senior Researcher CWI, from October 2009]

Administrative Assistant

Susanne van Dam [Secretary CWI, HdR]

2. Overall Objectives

2.1. Presentation

Software is still 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, software systems are these days 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 Each system is large, uses incompatible — new, outdated or non-standard — technology and above all, exhibits failures.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 phletora 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[1] Firstly, we study fact extraction: to extract information from existing software systems. This extracted information is vital to construct sound abstract models that can be used in further analysis (such as model checking or static analysis). Automated fact extraction is still expensive and error-prone.

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.

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 and Digital Forensics to experiment with this subject.

2.2. Highlights of the year

Release In October we completed the first version of the Rascal language that runs completely on the Java Virtual Machine. Rascal is open-source (BSD4) licensed. This release represents several innovative aspects of meta programming language research [22]. It was used immediately in the Software Evolution course at Universiteit van Amsterdam.

Book Publication of the book "Computational Semantics with Functional Programming" [29].

Best PhD abstract Best student research abstract award SLE 2010 Doctoral Symposium for Bas Basten [14].

3. Scientific Foundations

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. Fact extraction

This research focuses on source code; to analyze it and to transform it. Each analysis or transformation begins with fact extraction.

[1] Our work on reliable middleware is currently on hold since key researchers in this subarea left the team.

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 want to explore is the use of syntax-related program patterns for fact extraction. An early proposal for such a pattern-based approach is described in: a fixed base language (either C or PL/1 variant) is extended 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 closures) and the performance problems encountered, this approach has not seen wider use.

Another approach is proposed by de Moor and colleagues and uses path expressions on the syntax tree to extract program facts and formulate queries on them. This approach builds on the work of Paige and attempts to solve a classic problem: how to incrementally update extracted program facts (relations) after the application of a program transformation.

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 real (legacy) programming languages, which is highly relevant for experimental research and validation.

3.2.1. Goals

The main goal is to replace labour-intensive manual programming of fact extractors by automatic generation from annotated grammars or other concise and formal notation. 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. We expect to develop new ideas and techniques for generic (language-parametric) fact extraction from source code and other software artifacts.

3.3. Relational paradigm

For any source code analysis or transformation, after fact extraction comes elaboration, aggregation or other further analyses of these facts. For fact analysis, we base our entire research on the simple formal concept of a "relation".

There are at least three lines of research that have explored the use of relations. First, in SQL, n-ary relations are used as basic data type and queries can be formulated to operate on them. SQL is widely used in database applications and a vast literature on query optimization is available. There are, however, some problems with SQL in the applications we envisage: (a) Representing facts about programs requires storing program fragments (e.g., tree-structured data) and that is not easy given the limited built-in datatypes of SQL; (b) SQL does not provide transitive closures, which are essential for computing many forms of derived information; (c) More generally, SQL does not provide fixed-point computations that help to solve sets of equations. Second, in Prolog, Horn clauses can be used to represent relational facts and inference rules for deriving new facts. Although the basic paradigm of Prolog is purely declarative, actual Prolog implementations add imperative features that increase the efficiency of Prolog programs but hide the declarative nature of the language. Extensions of Prolog with recursion have resulted in Datalog in many variations [AHV95]. In F(p)–L a Prolog database and a special-purpose language are used to represent and query program facts.

Third, in SETL, the basic data type was the set. Since relations can easily be represented as sets of tuples, relation-based computations can, in principle, be expressed in SETL. However, SETL as a language was very complicated and has not survived. However, work on programming with sets, bags and lists has continued well into the 90's and has found a renewed interested with the revival of Lisp dialects in 2008 and 2009.

We have already mentioned the relational program representation by Linton. In Rigi, a tuple format (RSF) is introduced to represent untyped relations and a language (RCL) to manipulate them. Relational algebra is used in GROK, Crocopat and Relation Partition Algebra (RPA) to represent basic facts about software systems and to query them. In GUPRO graphs are used to represent programs and to query them. Relations have also been proposed for software manufacture, software knowledge management, and program slicing. Sometimes, set constraints are used for program analysis and type inference. More recently, we have carried out promising experiments in which the relational approach is applied to problems in software analysis and feature analysis. Typed relations can be used to decouple extraction, analysis and visualization of source code artifacts. These experiments confirm the relevance and viability of the relational approach to software analysis, and also indicate a certain urgency of the research direction of this team.

3.3.1. Goals

- New ideas and techniques for the efficient implementation of a relation-based specification formalism.
- Design and prototype implementation of a relation-based specification language that supports the use of extracted facts (Rascal).
- We target at uniform reformulations of existing techniques and algorithms for software analysis as well as the development of new techniques using the relational paradigm.

3.4. Refactoring and Transformation

The final 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.

3.4.1. 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.

4. Application Domains

4.1. Software Asset Management

Software represents major long term investments for most industries, including and perhaps most importantly the public sector. The cost of constructing and maintaining software are notoriously high. Our research results and research prototype tools are applicable to mitigate these costs. As opposed to outsourcing valuable and critical business information, we focus on a higher effectivity “at home”.

The application of source code analysis and transformation techniques is found in:

- Reverse engineering - reconstructing architectural overviews and metrics from source code to be able to do change impact analysis, risk analysis or re-design.
- Reengineering - large scale automated restructuring of source code.
- Refactoring - small scale, step-by-step, quality improvement of source code.

These applications help to improve the software construction and maintenance process. They can be supported by source code analysis and transformation tools, but only if appropriately flexible and comprehensible methods exist to construct them.

4.2. Domain Specific Languages

Another application of source code analysis and transformation is the construction of compilers for Domain Specific Languages (DSLs). Businesses struggle with the high rate of change to wanted functionality of software (requirements). A good example is the public sector, with its evolving laws and regulations. The construction of so called "Domain Models", which capture the fixed and the variable concepts of a business domain, and based on that the construction of a DSL promises to mitigate the cost of an "every changing environment" in software engineering.

A DSL compiler is based on analysis and transformation of a formal notation, just as normal programming language compilers are. Firstly, the difference from normal compilers are that there are less resources (time and money) to invest. Secondly, the scope of the language is smaller, and thus also more subject to change in requirements. Again, flexible and comprehensible methods for source code analysis and transformations should mitigate the cost of developing and maintaining these tools.

5. Software

5.1. AmbiDexter

Participant: Bas Basten [correspondent].

AmbiDexter is the tool written by Bas Basten to statically detect ambiguity in context-free grammars. It is the result of his research in novel algorithms for ambiguity detection that produce results quickly and informatively.

This tool is used to validate the usability and scalability of the newly published algorithms. It is made available in open-source and is in constant development when new experimental algorithms are added to it.

5.2. Derrick

Participants: Tijs van der Storm, Jeroen van den Bos [correspondent].

Derrick is a Domain Specific Language developed by Jeroen van den Bos. This language is an experiment in DSL design that tries to encapsulate all the variability in the construction of so-called "carving" algorithms. Such algorithms recover information that has been deleted or otherwise scrambled on digital media such as harddisks, usb sticks and mobile phones.

This tool is implemented using the Rascal language (see below) and will be made available in open-source after validation on real life digital forensics cases.

5.3. Rascal

Participants: Paul Klint, Jurgen Vinju [correspondent], Tijs van der Storm, Bas Basten, Jeroen van den Bos, Mark Hills, Bert Lisser, Arnold Lankamp.

5.3.1. Description

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.

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.

5.3.2. New technical features

The development of Rascal has started in 2009 and was continued in 2010, leading to an alpha release in October, which represents:

- A fully integrated syntax definition formalism
- A parser generator for general scannerless top-down parsers (based on GLL)
- An AST code generator that takes the previous as input (new)
- A fully Java based interpreter for Rascal (new)
- A type checker for Rascal, written in Rascal (new)
- A Rascal implementation of the Box language for source code formatting
- An Eclipse-based IDE for Rascal, featuring module management, syntax highlighting, outline, and an interactive scripting console
- Libraries for Java fact extraction
- Generic Libraries for Source Code (Version) Repository Mining (new)
- Generic libraries for visualization (new)

5.3.3. Experimental applications

- GLL parser generator (new)
- Rascal type checker (new)
- Implementation of Derrick, DSL for Digital forensics (new)
- Static analyses for PHP4 to PHP5 migration (new, in progress)
- General Source Repository Mining library (new)

- General Source code visualization library (new)
- Interactive (online) Rascal tutor environment (new)
- “Infer generic type arguments” refactoring for Featherweight Java
- LR(1) parse table generator
- Java/Eclipse JDT AST bindings and analyses
- Implementation of the Waebric DSL for XHTML generation
- etc.

5.3.4. Contributions and Documentation

- The last alpha release of Rascal is now independent of any C code, and runs purely on the Java virtual machine. This facilitates cooperation with Universiteit van Amsterdam and IBM TJ Watson Research institute, Eclipse.org and The Software Improvement Group.
- Rascal has a fully informative user-manual online at http://www.rascal-mpl.org
- An interactive tutor environment was developed for aiding in the teaching of Rascal. This environment can be deployed in Eclipse, as well as online on the web.
- Rascal was taught at the Universiteit van Amsterdam during the course Software Evolution for the Master Software Engineering (see above)

5.4. IDE Meta-tooling Platform

Participants: Jurgen Vinju [correspondent], Arnold Lankamp.

IMP, the IDE meta tooling platform is an Eclipse plugin developed mainly by the team of Robert M. Fuhrer at IBM TJ Watson Research institute. It is both an abstract layer for Eclipse, allowing rapid development of Eclipse based IDEs for programming languages, and a collection of meta programming tools for generating source code analysis and transformation tools.

Jurgen Vinju and Arnold Lankamp contribute significantly to the development of IMP. Their effort was focused on the maintenance and optimization of a general purpose symbolic representation library for source code artifacts, called “PDB”. PDB stands for Program DataBase. For more information, please visit http://www.eclipse.org/imp.

5.5. ASF+SDF Meta-Environment

Participants: Jurgen Vinju [correspondent], Paul Klint, Tijs van der Storm, Arnold Lankamp.

The ASF+SDF Meta-Environment is a long standing (over 10 years) research product and software laboratory developed by the researchers of ATEAMS. It is currently in servicing and maintenance stage, where it is applied in industrial settings and application research settings. However, it is currently not on the active list of research goals for ATEAMS. Instead the focus lies with the Rascal experiment.

6. New Results

6.1. Ambiguity Detection in Context-free Grammars

We can now effectively detect and report ambiguity in many grammars for real-life programming languages, such as Java, C and Cobol. This is due to the algorithmic innovations by Bas Basten [17], [16], [15], [14].

The next step is to scale ambiguity detection to so-called scannerless parsers. Scannerless parsers are used to parse programming languages with complex lexical syntax (often seen in legacy languages).

6.2. Domain Specific Language for Meta Programming

We have designed and implemented Rascal, which will serve as our laboratory environment for constructing many different kinds of software analyses, transformations and generators. This year the alpha version was released which enables many of such applications already [22].

It encouraging to see that Master Students with no formal background are able to construct reverse engineering tools and software metric tools using Rascal within one day of introductory teaching.

6.3. Domain Specific Language for Digital Forensics

Jeroen van den Bos has constructed this DSL [18] that is able to generate implementations of carving algorithms that compete with hand written ones. They are more precise and they are fully generated and thus easily adapted to new specialized circumstances. It is very common in digital forensics to manually adapt carving algorithms for special circumstances (hardware, media, file systems, file formats).

6.4. Epistemic Modeling for Protocol Analysis

The PhD thesis of Yanjing Wang represents a wide and deep investigation in the application of epistemic modeling for protocol analysis [8].

7. Contracts and Grants with Industry

7.1. UvA

- Paul Klint is employed by Universiteit van Amsterdam for 0.4fte for directing the Master Software Engineering.
- Jan van Eijck is contracted by Universiteit van Amsterdam for 0.2fte, for teaching the course Software Testing.
- Tijs van der Storm is contracted by Universiteit van Amsterdam for 0.2fte, for teaching the courses Software Evolution and Software Construction.
- Jurgen Vinju is contracted by Universiteit van Amsterdam for 0.2fte, for teaching the courses Software Evolution and Software Construction.

7.2. National Initiatives

+ VEMPS, VeriÞcation and Epistemics of Multi-Party Protocol Security, Funding “Open Competitive” NWO (NWO project funding) Partners are Technische Universiteit Eindhoven, Vrije Universiteit, Uni versiteit Leiden and Universiteit Utrecht.
+ NWO TOP proposal “Domain-Specific Languages: A Big Future for Small Programs”
+ “Next Generation Auditing: Data Assurance as a Service” (Jacquard project)

8. Other Grants and Activities

8.1. Exterior research visits

The project had the following visitors during the year 2010:

- Dr. Lakshmanan Kuppusami (May—June 2010)
- Anya Helene Bagge (Institutt for Informatikk Universitetet i Bergen, one year starting October 2010)

Furthermore, our team members visited other institutes in 2010:

- Tijs van der Storm visited Semantic Designs, U.S.A.
- Tijs van der Storm visited University of Texas, U.S.A.
- Bas Basten visited Bergen Language Design Laboratory of University of Bergen in Norway (September 19-26, 2010).
- Paul Klint visited Bergen Language Design Laboratory of University of Bergen in Norway, Opening of BLDL, (November 4, 2010)

9. Dissemination

9.1. Services to the scientific community

9.1.1. Research Management

- (Jan van Eijck) Member of the European Network in Computational Logic (initiated by the ESPRIT Basic Research Action 'Compulog'), since March 1997.
- (Jan van Eijck) Member of the international quality assessment committee for Computer Science and AI in Flanders, Belgium (since January 2009).
- (Jan van Eijck) Coordinator of NWO Project VEMPS (since June 2006).
- (Paul Klint) Head of Software engineering Department, CWI.
- (Paul Klint) Head of CWI Research group Software Analysis and Transformation (SEN1)
- (Paul Klint) Visiting Professor University of London, Royal Holloway.
- (Paul Klint) Project leader ATEAMS (CWI/INRIA joint project)
- (Paul Klint) Full Professor at UvA, Software Engineering Chair.
- (Paul Klint) Treasurer European Association for Programming Languages and Systems (EAPLS).
- (Paul Klint) Member of the TWINS Council (Royal Dutch Academy of Science)
- (Paul Klint) Member ETAPS steering committee,
- (Paul Klint) Scientific council Lorentz Institute,
- (Paul Klint) Board Instituut voor Programmatuur en Architectuur (IPA)

9.1.2. Participation to Working Groups

- (Paul Klint) Software EngineeRing for rEsilieNt systEms (SERENE)
- (Paul Klint) ERCIM working group Rapid Integration of Software Engineering (RISE),
- (Paul Klint) ERCIM working group Software Evolution (EVOL)

9.1.3. Organizations of sessions, workshops and conferences

- (Jurgen Vinju) organizing chair of Language Descriptions Tools and Applications (LDTA 2010)
- (Jurgen Vinju) program committee co-chair of the International Working Conference on Source Code Analysis and Manipulation (SCAM 2010)
- (Jurgen Vinju) steering committee member of the International Working Conference on Source Code Analysis and Manipulation
- (Jurgen Vinju) steering committee member of the International Conference on Software Language Engineering (SLE)

- (Yanjing Wang) organizer of The Many Faces of Protocols and Knowledge, organized after his PhD defense in September 21st 2010.

9.1.4. Editorial boards

- (Paul Klint) Editor for Science of Computer Programming
- (Paul Klint) Editor for Springer Service Science Book Series
- (Jurgen Vinju) Co-editor of the special issue of Science of Computer Programming on Language Descriptions Tools and Applications (to appear)
- (Jurgen Vinju) Co-editor of the special issue of Science of Computer Programming on Source Code Analysis and Manipulation (in preparation)

9.1.5. Reviews

- Review of journal papers:
  - (Paul Klint, Tijs van der Storm) IEEE Transactions on Software Engineering
  - (Mark Hills) Journal of Logic and Algebraic Programming
- Review of research projects:
  - (Jan van Eijck) Member of the international quality assessment committee for Computer Science and AI in Flanders, Belgium.
  - (Paul Klint) Member of the programme committees for NWO Jacquard Programme
  - (Jurgen Vinju) Member of the programme committee for NWO Vernieuwingsimpuls Programme
  - (Tijs van der Storm) Chair of Internship Committee CWI
  - (Paul Klint) ICTRegie jury member
  - (Paul Klint) Member evaluation panel, Portuguese Science and Technology Foundation.
  - (Jurgen Vinju) Member evaluation panel, Ontario Centres of Excellence (Canada)

9.1.6. Program Committees

- (Jurgen Vinju, Tijs van der Storm) International Working Conference on Source Code Analysis and Manipulation (SCAM)
- (Jurgen Vinju) Software Language Engineering (SLE)
- (Paul Klint, Jurgen Vinju) Software Language Engineering Doctoral Symposium (SLE)
- (Jurgen Vinju) International Workshop on Academic Software Development Tools and Techniques (Wasdett)
- (Jurgen Vinju) ACM Symposium on Applied Computing (SAC)
- (Paul Klint) Automated Software Engineering 2010 (ASE10)
- (Paul Klint) Benevol 2010
- (Paul Klint) Software Composition 2010
- (Jan van Eijck) International Conference on Computational Semantics (IWCS 2011)

9.1.7. Invited talks

- (Paul Klint) EASY Meta-Programming with Rascal, Open Unversity, Heerlen, Netherlands, March 5
- (Paul Klint) New Programming Methods, ICT Delta, New Programming Methods, Utrecht, Netherlands, March 18.

- (Paul Klint) EASY Meta-programming with Rascal, Dagstuhl Seminar on Relationships, Objects, Roles and Queries in Modern programming Languages, Dagstuhl, Germany, April 11-16
- (Paul Klint) Using Grammars and Relations to Formalize Models, Keynote, ECOOP Workshop Formalization of Modeling Languages, Maribor, Slovenia, June 21.
- (Paul Klint) Building Academic Software Tools: Do's and Don'ts, Keynote WASDeTT Workshop at Automated Software Engineering, Antwerp, Belgium, september 20.
- (Paul Klint) Towards Visual Software Analytics, ADAPT Summerschool September 27-October 1, Koblenz, Germany.
- (Paul Klint) Gold Digging in Software, Sciencepark Amsterdam, Open Day, Amsterdam, Netherlands, October 9.
- (Jurgen Vinju) Rascal - open-source research software for improving any other software, Software Freedom Day. Sciencepark Amsterdam, Netherlands, September 17.
- (Jurgen Vinju) Rascal - An instrument for software research. CWI Scientific Meeting, February 2010.
- (Jurgen Vinju) Software Research CWI, for the CWI Scientific Advisory Committee, Amsterdam, Netherlands, November 5, 2010.
- (Jurgen Vinju) Rascal Metaprogramming Made EASY, FIRST, IT-Universitetet København (Danmark), February 18th, 2010
- (Jan van Eijck) Social Software, Keynote, It's a Logic World, Incogito Lustrum, January 8, 2010.
- (Jan van Eijck) How to Verify an Epistemic Protocol with DEL, Lorentz workshop on Formal Theories of Communication, Leiden, February 22, 2010.

9.1.8. Phd and MSc committees

- Master Thesis Software Engineering, Universiteit van Amsterdam (Tijs van der Storm, Jurgen Vinju, Paul Klint, Jan van Eijck)
  - Chris Brouwer, Coupled transformations to automate changes to co- evolving artifacts.
  - Jelle Geelhoed, A comparison of transformation languages and their capability to produce a maintainable compiler.
  - Stephan J. Preeker, Comparing DSL language implementation in: Python, Java, JavaScript and C#.
  - Timon Snetselaar, Better maintainability through Event-Driven Architecture.
  - Arseni Storojev, Extraction of topic cloud sets from collections of text documents and spreadsheets.
  - Taco Witte, Assessing the impact of API evolution. An experiment with C++ and the Qt framework.
  - David Walschots, A case study on the cost and benefits of bus-oriented architectures.
  - Waruzjan Shahbazian, Rminer: An integrated model for repository mining using Rascal.
  - Nico Schoenmaker, Over de understandability van subtype polymorfisme in objectgeoriënteerde systemen.
  - Steven Raemakers, Testing Candidates For Semantic Clone Detection.
  - Maarten Wullink, Data model Maintainability - A comparative study of maintainability metrics.
  - Sander Vellinga, Identifying behavior changes after PHP language migration using static source-code analysis.
  - Tigran Kalaidjan, Specialist in Car.

  - Bas Slagter, De effecten van Contract Driven Development In de praktijk.
- Phd Committees (Paul Klint)
  - Cathal Boogerd, TU Delft, january 19, 2010.
  - Diego Ordonez, Louvain-la-Neuve, May 1 and Augustus 26, 2010.
  - Vadim Zaytsev, Vrije Universiteit, Amsterdam, October 27, 2010
- Phd Committees (Jan van Eijck)
  - Yanjing Wang, Universiteit van Amsterdam, September 23, 2010, (Supervisor)
  - Christina Unger, Universiteit Utrecht, March 31, 2010, (Supervisor)
  - Eline Westerhout, Universiteit Utrecht, June 2, 2010, (Committee member)
- MSc Committees (Jan van Eijck)
  - Jeroen Bransen, Universiteit Utrecht, CKI.
  - Matthew Wampler Doty, Universiteit van Amsterdam, ILLC.
  - Remi Turk, Universiteit van Amsterdam, ILLC.

9.2. Teaching

- (Paul Klint, Jurgen Vinju and Tijs van der Storm) 'Software Construction', Course for Master students of Software Engineering, Universiteit van Amsterdam, Januari-March.
- (Paul Klint, Jurgen Vinju and Tijs van der Storm) 'Software Evolution', Course for Master students of Software Engineering, UvA, Oktober-December.
- (Jan van Eijck and Floor Sietsma) 'Software Testing', Course for Master Students of Software Engineering, Amsterdam, September-November.
- (Jurgen Vinju, Bas Basten, Mark Hills, Tijs van der Storm) Rascal, 5 languages Summerschool, Universiteit van Amsterdam, Netherlands, July 14, 2010
- (Jan van Eijck) Verzamelingen, Lijsten, Functioneel Programmeren, Lecture for Pupils Ignatius College Amsterdam, May 17.
- (Jan van Eijck) Haskell: Programming in a Lazy, Purely Functional Language', 5 Languages Summerschool, Amsterdam, June 12 2010.
- (Jurgen Vinju) Rascal Codefest, Devnology.org, TTY, Amsterdam, Netherlands, March 3, 2010.

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, p. 76-90.

[2] B. BASTEN, J. VINJU. *Faster Ambiguity Detection by Grammar Filtering*, in "Proceedings of the tenth workshop on Language Descriptions Tools and Applications", C. BRABRAND, P.-E. MOREAU (editors), 2010.

[3] G. ECONOMOPOULOS, P. KLINT, J. VINJU. *Faster Scannerless GLR Parsing*, in "CC '09: Proceedings of the 18th International Conference on Compiler Construction", Berlin, Heidelberg, Springer-Verlag, 2009, p. 126–141.

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

[5] P. KLINT, T. VAN DER STORM, J. VINJU. *RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation*, in "Source Code Analysis and Manipulation, IEEE International Workshop on", Los Alamitos, CA, USA, 2009, p. 168-177, http://doi.ieeecomputersociety.org/10.1109/SCAM.2009.28.

[6] P. KLINT, T. VAN DER STORM, J. VINJU. *EASY Meta-Programming with RASCAL*, in "Generative and Transformational Techniques in Software Engineerin", J. FERNANDES, R. LAEMMEL, J. SARAIVA, J. VISSER (editors), Lecture Notes in Computer Science, Springer, Heidelberg, 2010, n^{o} 6491, p. 222–298, to appear.

[7] Y. WANG, F. SIETSMA, JAN VAN. EIJCK. *Composing Models*, in "9th Conference on Logic and the Foundations of Game and Decision Theory", July 2010, to appear.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[8] Y. WANG. *Epistemic Modelling and Protocol Dynamics*, Universiteit van Amsterdam, 2010.

Articles in International Peer-Reviewed Journal

[9] D. O. CAMACHO, K. MENS, M. VAN DEN BRAND, J. VINJU. *Automated generation of program translation and verification tools using annotated grammars*, in "Science of Computer Programming", 2010, vol. 75, n^{o} 1-2, p. 3-20, http://dx.doi.org/10.1016/j.scico.2009.10.003.

[10] F. DECHESNE, Y. WANG. *To know or not to know: epistemic approaches to security protocol verification*, in "Synthese", 2010, p. 1-26, 10.1007/s11229-010-9765-8.

[11] H. V. DITMARSCH, J. VAN EIJCK, W. WU. *Verifying one hundred prisoners and a lightbulb*, in "Journal of Applied Non-classical Logics", 2010, to appear.

[12] J. VAN EIJCK. *The language of social software*, in "Synthese", 2010, p. 1-20, http://dx.doi.org/10.1007/s11229-010-9766-7.

International Peer-Reviewed Conference/Proceedings

[13] T. T. BARTOLOMEI, K. CZARNECKI, R. LAEMMEL, TIJS VAN DER. STORM. *Study of an API migration for two XML APIs*, in "Postproceedings of Software Language Engineering (SLE 2009)", LNCS, Springer, 2010.

[14] B. BASTEN. *Practical Ambiguity Detection for Context-Free Grammars*, in "Doctoral Symposium of the International Conference on Software Language Engineering", 2010.

[15] 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, p. 76-90.

[16] B. BASTEN, TIJS VAN DER. STORM. *AmbiDexter: Practical Ambiguity Detection, Tool Demonstration*, in "Proceedings of the Tenth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'10)", J. VINJU, C. MARINESCU, P. CUOQ (editors), IEEE, 2010, to appear.

[17] B. BASTEN, J. VINJU. *Faster Ambiguity Detection by Grammar Filtering*, in "Proceedings of the tenth workshop on Language Descriptions Tools and Applications", C. BRABRAND, P.-E. MOREAU (editors), 2010.

[18] JEROEN VAN DEN. BOS. *Domain-Specific Languages for Digital Forensics*, in "Doctoral Symposium of the International Conference on Software Language Engineering", 2010.

[19] H. V. DITMARSCH, J. VAN EIJCK, W. WU. *One hunderd prisoners and a lightbulb — logic and computation*, in "Proceedings of KR 2010 Toronto", F. LIN, U. SATTLER (editors), AAAI Press, 2010, to appear.

[20] M. HILLS, G. ROSU. *A Rewriting Logic Semantics Approach to Modular Program Analysis*, in "Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010", Edinburgh, Scottland, UK, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, July 11-13 2010, vol. 6, p. 151-160, http://dx.doi.org/10.4230/LIPIcs.RTA.2010.151.

[21] P. KLINT, TIJS VAN DER. STORM, J. VINJU. *On the Impact of DSL tools on the Maintainability of Language Implementations.*, in "Proceedings of the tenth workshop on Language Descriptions Tools and Applications", 2010, to appear.

[22] P. KLINT, T. VAN DER STORM, J. VINJU. *EASY Meta-Programming with RASCAL*, in "Generative and Transformational Techniques in Software Engineerin", J. FERNANDES, R. LAEMMEL, J. SARAIVA, J. VISSER (editors), Lecture Notes in Computer Science, Springer, Heidelberg, 2010, n^{o} 6491, p. 222–298, to appear.

[23] V. LUSSENBURG, TIJS VAN DER. STORM, J. VINJU, J. WARMER. *Mod4J: A Qualitative Case Study of Model-Driven Software Development*, in "13th International Conference on Model Driven Engineering Languages and Systems", D. PETRIU, N. ROUQUETTE, O. HAUGEN (editors), Lecture Notes in Computer Science, Springer, October 2010.

[24] Y. WANG, JAN VAN. EIJCK, HANS VAN. DITMARSCH. *On the Logic of Lying*, in "International Workshop on Games, action and social software", J. V. EIJCK, R. VERBRUGGE (editors), 2010, to appear.

[25] Y. WANG, F. SIETSMA, JAN VAN. EIJCK. *Composing Models*, in "9th Conference on Logic and the Foundations of Game and Decision Theory", July 2010, to appear.

[26] Y. WANG, F. SIETSMA, J. VAN EIJCK. *Logic of information flow on communication channels*, in "Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1", Richland, SC, AAMAS '10, International Foundation for Autonomous Agents and Multiagent Systems, 2010, p. 1447–1448, http://portal.acm.org/citation.cfm?id=1838206.1838425.

[27] S. DE GOUW, F. DE BOER, J. VINJU. *Prototyping a tool environment for run-time assertion checking in JML with Communication Histories*, in "12th Workshop on Formal Techniques for Java-like Programs", 2010, to appear.

[28] T. VAN DER STORM. *Relational Meta-Modeling*, in "Relationships, Objects, Roles, and Queries in Modern Programming Languages", G. BOELLA, E. MEIJER, D. J. PEARCE, F. STEIMANN, F. TIP (editors), Dagstuhl Seminar Proceedings, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010, n^{o} 10152, Abstract.

Scientific Books (or Scientific Book chapters)

[29] JAN VAN. EIJCK, C. UNGER. *Computational Semantics with Functional Programming*, Cambridge University Press, September 2010.

[30] J. V. EIJCK, H. KAMP. *Discourse Representation in Context*, in "Handbook of Logic and Language", J. V. BENTHEM, A. TER MEULEN (editors), Elsevier, 2010, to appear.