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

1. Team

Research Scientist

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]
Yaroslav Usenko [Senior Researcher CWI]
Emilie Balland [Visiting scientist CWI]

Technical Staff

Bert Lisser [Scientific Programmer CWI, HdR]
Arnold Lankamp [Scientific Programmer CWI]
Michael Smeding [Technical support CWI, HdR]

PhD Student

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

Over the last decades, computer science has delivered various insights how to organize software. 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 Each system is large, uses incompatible — new, outdated or non-standard — technology and above all, exhibits failures.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 heterogeneous and large software systems and to refactor and transform them in order to keep them up-to-date. With the phletora of languages and technology platforms it is becoming very difficult and very expensive to construct tools to achieve this.

The main challenge of this project is to address this problem by (a) developing language-parametric techniques for analysis and transformation that are (b) embedded in an scalable and reliable tool infrastructure. We will use the service paradigm itself to build better (better quality, better flexibility and better performance) tools for analysis and transformation of software systems. To this end, we will study fact extraction (to extract information from existing software systems), refactoring and transformation (to improve them), and reliable middleware to act as tool composition foundation.

The innovative aspects of this project are twofold: fact extraction, refactoring and transformation are approached in a single conceptual framework and service-orientation will be used as the basis for their implementation. In this way tool compositions can be validated and software workbenches and massively parallel services can be created that open up new horizons for software understanding and transformation. The breadth, depth and volume of software that can be analyzed and improved in this way is significant.

2.2. Highlights of the year

Please note that the ATeams project started in July 2009.

Start of ATEAMS The official start of ATEAMS at CWI was in July 2009. The occasion was celebrated in Amsterdam, in the presence of M. François Delahousse (Premier conseiller de l'ambassade de France aux Pays-Bas), Michel Cosnard (President INRIA), Jean-Pierre Banâtre (Director of European Partnerships INRIA), Jan Karel Lenstra (Director CWI), and Jos Engelen (Chairman Dutch Organisation for Scientific Research, NWO).

GTTSE Summer School Paul Klint, Tijs van der Storm and Jurgen Vinju presented a course in Source Code Analysis and Transformation using the newly developed Domain Specific Language "Rascal" at the International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE) on 6Ð11 July, 2009, Braga, Portugal

Software Evolution Course Paul Klint, Tijs van der Storm and Jurgen Vinju started teaching the Rascal DSL at the Universiteit van Amsterdam in October 2009. This represents a first evaluative step in validating the language design for usability by non-expert programmers.

Visitor Emilie Balland visited us for two times two months with significant contributions to ATEAMS

New Phd Jeroen van den Bos arrived in September to start on a domain specific language for digital forensics (co-funded by Dutch National Forensics Institute).

New post-doc Mark Hills arrived in October 2009 to strengthen ATEAMS.

Keynote Paul Klint P. Klint, Grammarware is everywhere and what to do about that? Keynote at GDR/GPL Toulouse, France.

Keynote Paul Klint Language Design for Meta-programming in the Software Composition Domain, Keynote at Software Composition, Zurich, Switserland.

Keynote Paul Klint Easy Meta-Programming with Rascal, Keynote at Opening of Bergen Language Design Center, Bergen Norway.

Keynote Jan van Eijck The Language of Social Software, Workshop on Logic and Social Interaction, Chennai, India

Visit Alcatel-Lucent Bell Labs Bas Basten visited Dennis Dams at Bell Labs for an internship on static analysis.

Visit Universität Koblenz-Landau Tijs van der Storm visited Prof. Dr. Ralf Laëmmel twice to work on research concerning API migration (a kind of large scale refactoring).

3. Scientific Foundations

3.1. Fact extraction

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

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.

3.1.1. 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.

3.2. 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.

3.2.1. 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.

fondements

3.3. 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.

3.3.1. 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. Reliable Middleware

As already sketched earlier, Service-Oriented Computing (SoC) forms the next evolutionary step in the creation of flexible, distributed, software systems. A key question is what is the best way to distribute business logic in a service-oriented architecture so that the services remain reusable and the process flow remains maintainable. In this project we scope these issues to the domain of distributing analysis and transformation tasks over different tools and making these tools cooperate.

3.4.1. Goals

The key question is to investigate how the realities of workflow-based solutions and the promises of service-oriented systems can be reconciled. Our goal is to obtain reliable tool composition can be achieved for scalable analysis and transformation systems.

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. Rascal

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

5.1.1. Description

The Rascal Domain Specific Language for Source code analysis and Transformation is developed by ATeams, mainly in 2009. 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.

5.1.2. New technical features

The development of Rascal has started in 2009, leading to:

- A context-free grammar for Rascal
- A AST code generator that takes the previous as input
- A full interpreter for Rascal
- A type checking algorithm (embedded in the interpreter)
- An Eclipse-based IDE for Rascal, featuring module management, syntax highlighting, outline, and an interactive scripting console

5.1.3. Experimental applications

- "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.

5.1.4. Contributions and Documentation

- Rascal has a fully informative user-manual online at http://www.meta-environment.org/Meta-Environment/Rascal
- Rascal was taught at the GTTSE summer school (see above)
- Rascal was taught at the Universiteit van Amsterdam during the course Software Evolution for the Master Software Engineering (see above)

5.2. 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 development 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.

5.3. 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. Contracts and Grants with Industry

6.1. UvA

- Paul Klint is contracted 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. Other Grants and Activities

7.1. National actions

- HEFBOOM project, Dutch National Research Institute (NWO) project for the dissemination of research results.
- VEMPS, VeriÞcation and Epistemics of Multi-Party Protocol Security, Funding "Open Competitie" NWO (NWO project funding) Partners are Technische Universiteit Eindhoven, Vrije Universiteit, Universiteit Leiden and Universiteit Utrecht.

7.2. Visits and researcher invitations

The project had the following visitors during the year 2009:

- Emilie Balland (INRIA-LORIA, two times two months)
- Claus Brabrand (IT University of Copenhagen, one week)
- Anya Helene Bagge (Institutt for Informatikk Universitetet i Bergen, one month)
- Lakshmanan Kuppusami (ERCIM Postdoc Fellow)

8. Dissemination

8.1. Services to the scientific community

8.1.1. 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 Interactive Software Maintenance and Renovation (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)

8.1.2. 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)

8.1.3. Organizations of sessions, workshops and conferences

- (Jurgen Vinju) co-chair of Language Descriptions Tools and Applications 2009
- (Jurgen Vinju) organizing chair of Language Descriptions Tools and Applications 2010

8.1.4. Editorial boards

- (Paul Klint) Editor for Science of Computer Programming
- (Paul Klint) Editor for Springer Service Science Book Series

8.1.5. Reviews

- Review of journal papers:
  - (Jurgen Vinju) Software Practise and Experience
  - (Jurgen Vinju, Paul Klint, Tijs van der Storm) IEEE Software
  - (Jurgen Vinju, Paul Klint) Science of Computer Programming (Special Issue on Programming Languages Track of the ACM SAC'08, Special issue on Experimental software and toolkits (EST), and WASDETT special issue)
  - (Paul Klint) IEEE Transactions on Software Engineering
  - (Paul Klint) Journal of Software Maintenance and Evolution: Research and Practice
  - (Yanying Wang) Journal of Applied Non-Classical Logics
- Review of research projects:
  - (Jan van Eijck) Member of the international quality assessment committee for Computer Science and AI in Flanders, Belgium (since January 2009).
  - (Paul Klint) Member of the programme committees for NWO Jacquard Programme,

8.1.6. Program Committees

- (Jan van Eijck) International Conference on Computational Semantics (IWCS).
- (Jurgen Vinju, Tijs van der Storm) Language Description Tools and Applications (LDTA)
- (Jurgen Vinju) Formal Methods (FM)
- (Jurgen Vinju) International Working Conference on Source Code Analysis and Manipulation (SCAM)
- (Jurgen Vinju) Software Language Engineering (SLE)

- (Paul Klint) European Software Engineering Conference (ESEC)
- (Paul Klint) IFIP TC2 Central and East European Conference on Software Engineering Techniques (CEE-SET)
- (Paul Klint) BElgian-NEtherlands software eVOLution workshop (BENEVOL)
- (Paul Klint) Software EngineeRing for rEsilieNt systEms (SERENE)

8.1.7. Phd and MSc committees

- (Jan van Eijck) Member of Ph.D. thesis committee of Alexey Rodrigues, Utrecht, May 20, 2009.
- (Jan van Eijck) Member of Ph.D. thesis committee of Andreas Witzel, Amsterdam, September 3, 2009.
- (Paul Klint) Bob Diertens (Universiteit van Amsterdam)
- (Paul Klint) Ali Mesbah (University of Delft)
- (Paul Klint) Fabian Groffen (Universiteit van Amsterdam)
- (Paul Klint) Alex. van Ballegooij (Universiteit van Amsterdam)
- Master Thesis Software Engineering, Universiteit van Amsterdam (Tijs van der Storm, Jurgen Vinju, Paul Klint, Jan van Eijck)
  - F. de Bruijn
  - M. Koesen
  - D.V. van Dijk
  - J.J.A.W. van Lieshout
  - B.F.L. Ector
  - V.Q.J. Lussenburg
  - J. Fransen
  - E.A. Pronk
  - M. de Graaf
  - L.A.C. de Ridder
  - N. Heirbaut
  - J.L. van Schagen
  - R.M. Hommels
  - D. Verhamme
  - R. Zandbergen

8.2. 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 Yanying Wang) ‘Software Testing’, Course for Master Students of Software Engineering, Amsterdam, September-November.
- (Paul Klint, Jurgen Vinju and Tijs van der Storm) ‘Source code analysis and Transformation with Rascal‘, 3rd Summer School on Generative and Transformational Techniques in Software Engineering 6Ð11 July, Braga, Portugal.

- (Jan van Eijck) Uil-OTS Course 'Language, Mathematics and Logic'.
- (Jan van Eijck) 'Computational Semantics', LOT Summer School Course, Leiden, June 15–19 (together with Christina Unger).
- (Jan van Eijck together with Rineke Verbrugge) 'Social Software', ESSLLI'09 Course, Bordeaux, July 20–24.
- (Jan van Eijck) 'Social Software in Four Examples', Lectures for Mathematics Teachers Holiday Course, Amsterdam, 21 August, and Eindhoven, 28 August.
- (Jan van Eijck) 'Sets, Lists and Functional Programming', Lecture for Excellent Students at the Amsterdam Ignatiuscollege, May 28.

9. Bibliography

Year Publications

Articles in International Peer-Reviewed Journal

[1] 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", 2009, http://dx.doi.org/10.1016/j.scico.2009.10.003.

International Peer-Reviewed Conference/Proceedings

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

[3] D. BOSNACKI, A. MATHIJSSEN, Y. S. USENKO. *Behavioural Analysis of an I^2C Linux Driver*, in "FMICS", M. ALPUENTE, B. COOK, C. JOUBERT (editors), Lecture Notes in Computer Science, vol. 5825, Springer, 2009, p. 205-206, http://dx.doi.org/10.1007/978-3-642-04570-7.

[4] P. CHARLES, R. M. FUHRER, STANLEY M. JR. SUTTON, E. DUESTERWALD, J. VINJU. *Accelerating the Creation of Customized, Language-Specific IDEs in Eclipse*, in "Proceedings of OOPSLA 2009", ACM Sigplan, 2009, to appear.

[5] 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.

[6] J. V. EIJCK, F. SIETSMA. *Multi-agent Belief Revision with Linked Plausibilities*, in "Logic and the Foundations of Game and Decision Theory (LOFT8)", G. BONANNO, B. LOEWE, W. VAN DER HOEK (editors), Texts in Logic and Games, 2009.

[7] P. KLINT, T. VAN DER STORM, J. VINJU. *EASY Meta-Programming with RASCAL*, in "Generative and Transformational Techniques in Software Engineerin", J. FERNANDES, R. LAMMEL, J. SARAIVA, J. VISSER (editors), Universidade do Minho, 2009, p. 185–238.

[8] 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.

[9] P. KLINT, J. VINJU, T. VAN DER STORM. *Language Design for Meta-programming in the Software Composition Domain*, in "Software Composition, 8th International Conference, SC 2009, Zurich, Switzerland, July 2-3, 2009. Proceedings", A. BERGEL, J. FABRY (editors), Lecture Notes in Computer Science, vol. 5634, Springer, 2009, p. 1-4.

[10] Y. WANG, L. KUPPUSAMY, J. VAN EIJCK. *Verifying Epistemic Protocols under Common Knowledge*, in "TARK '09: Proceedings of the 12th Conference on Theoretical Aspects of Rationality and Knowledge", ACM (editor), 2009, p. 257–266.

Scientific Books (or Scientific Book chapters)

[11] M. ALPUENTE, B. COOK, C. JOUBERT (editors). *Formal Methods for Industrial Critical Systems, 14th International Workshop, FMICS 2009, Eindhoven, The Netherlands, November 2-3, 2009. Proceedings*, Lecture Notes in Computer Science, vol. 5825, Springer, 2009, http://dx.doi.org/10.1007/978-3-642-04570-7.

[12] J. V. EIJCK, R. VERBRUGGE (editors). *Discourses on Social Software*, Texts in Logic and Games, vol. 5, Amsterdam University Press, Amsterdam, 2009.

[13] J. V. BENTHEM, J. VAN EIJCK. *Game Theory, Logic and Rational Choice*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 123–134.

[14] F. DECHESNE, J. VAN EIJCK, W. TEEPE, Y. WANG. *What is Protocol Analysis?*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 135–146.

[15] F. DECHESNE, J. VAN EIJCK, W. TEEPE, Y. WANG. *Dynamic Epistemic Logic for Protocol Analysis*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 147–161.

[16] H. V. DITMARSCH, J. VAN EIJCK, R. VERBRUGGE. *Common Knowledge and Common Belief*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 99–121.

[17] J. V. EIJCK, C. UNGER. *Computational Semantics with Functional Programming*, Cambridge University Press, 2009, To appear.

[18] J. V. EIJCK, A. VISSER. *Dynamic Semantics*, in "The Stanford Encyclopedia of Philosophy", Stanford University, 2009.

[19] P. KLINT. *From Centaur to the Meta-Environment: a tribute to a great meta-technologist*, Cambridge University Press, 2009.

[20] J. VAN EIJCK, R. PARIKH. *What is Social Software?*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 37–47.

[21] J. VAN EIJCK, M. VAN HEES. *Ends and Means, Values and Virtues*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 87–98.

[22] J. VAN EIJCK. *A Guest Lecture on Social Software*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 49–56.

[23] J. VAN EIJCK. *On Social Choice Theory*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 71–85.

[24] J. VAN EIJCK, R. VERBRUGGE. *Social Software and the Ills of Society*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 219–226.

[25] J. VAN EIJCK, R. VERBRUGGE. *On Collective Rational Action*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 199–216.

[26] J. VAN EIJCK, R. VERBRUGGE. *Eating from the Tree of Ignorance*, in "Discourses on Social Software", Amsterdam University Press, 2009, p. 183–198.