IN PARTNERSHIP WITH:
Centrum Wiskunde & Informatica

Activity Report 2011

Project-Team ATEAMS

Analysis and Transformation based on rEliAble tool coMpositionS

RESEARCH CENTER
Lille - Nord Europe

THEME
Programs, Verification and Proofs

Table of contents

Project-Team ATEAMS

Keywords: Programming Languages, Formal Methods, Domain-Specific Languages, Software Engineering, Meta-modeling

ATEAMS is a joint research team with SEN1 of Centrum Wiskunde & Informatica (CWI) in Amsterdam, The Netherlands. 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. The research results of ATEAMS/SEN1 appear in yearly reports of both CWI and INRIA alike.

ATEAMS/SEN1 has merged with another group from CWI (INS3) in 2011. The results of this part of our team are not included in the current report because they are not related to the research topic of ATEAMS/SEN1.

1. Members

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]
Mark Hills [Post-doc CWI]
Anya Helene Bagge [Research Visitor Universitetet i Bergen, Norway]
Michael Godfrey [Research Visitor University of Waterloo, Canada]
Sunil Simon [Post-doc CWI]
Vadim Zaytsev [Post-doc CWI]

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]
Atze van der Ploeg [Junior Researcher CWI]
Davy Landman [Junior Researcher CWI]
Paul Griffioen [Junior Researcher CWI]
Anastasia Izmaylova [Junior Researcher CWI]
Jeroen van den Bos [Junior Researcher CWI]
Riemer van Rozen [Researcher Hogeschool van Amsterdam]
Floor Sietsma [Junior Researcher CWI]

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. 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 and Digital Forensics to experiment with this subject. In Computational Auditing we are focusing on type system extensions and in Digital Forensics on optimization and specialization.

2.2. Highlights

Professorship Jan van Eijck was appointed Professor at Universiteit van Amsterdam.

Distinguised scientist In August Michael Godfrey joined us for a year of his sabbatical from University of Waterloo (Canada).

Rascal release A new version of the Rascal meta-programming language was released in October.

Forensics The Derric domain specific language generates faster and more accurate implementations than its competitors in the digital forensics arena, using less than 1500 lines of code.

Inception of Ensō In a collaboration with University of Texas, Tijs van der Storm introduced the Ensō framework: a new form of programming based on the concepts of model driven engineering.

Eclipse The source code of Rascal was approved for inclusion in the Eclipse IMP source code and releases.

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. Software analysis

This research focuses on source code; to analyze it and to transform it. Each analysis or transformation begins with fact extraction. After the we may analyze specific software systems or large bodies of software systems. Our goal is to improve software systems by learning to understand the causes of complexity.

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.

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. 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.
- We apply the above in the reformulation of refactorings for Java and domain specific languages.

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.

3.5. 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.

3.5.1. 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.

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.

We currently focus researching applications of DSLs in the areas of Computational Auditing and Digital Forensics.

5. Software

5.1. AmbiDexter

Participants: Bas Basten [correspondent], Jurgen Vinju.

Characterization: A-3-up4, SO-4, SM-2-up3, EM-2-up3, SDL-4-up5, OC-DA-3-CD-3-MS-3-TPM-3.

WWW: http://homepages.cwi.nl/~basten/ambiguity/

Objective Statically detect ambiguity of context-free grammars for programming languages, as fast and precise as possible.

Users: Authors of context-free grammars of programming languages in SDF2, Rascal, ANTLR, etc

Impact: This is the first usable ambiguity detection tool, aiming to solve the Achilles' heal of context-free general parsing.

Competition: AmbiDexter is the fastest and most accurate tool currently available.

Engineering: AmbiDexter was developed by one person and will be maintained by another. It is 25 LOC in Java and distributed as a component of the Rascal IDE.

Publications: [14], [9] [2], [1]

5.2. Derric

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

Characterization: A-2-up3, SO-4, SM-2-up3, EM-3, SDL-3-up4, OC-DA-3-CD-3-MS-3-TPM-3.

WWW: http://svn.rascal-mpl.org/derric/

Objective: Encapsulate all the variability in the construction of so-called "carving" algorithms, then generate the fastest and most accurate implementations. Carving algorithms recover information that has been deleted or otherwise scrambled on digital media such as hard-disks, usb sticks and mobile phones.

Users: Digital forensic investigation specialists

Impact: Derric has the potential of revolutionizing the carving area. It does in 1500 lines of code what other systems need tens of thousands of lines for with the same accuracy. Derric will be an enabler for faster, more specialized and more successful location of important evidence material.

Competition: Derric competes in a small market of specialized open-source and commercial carving tools.

Engineering: Derric is a Rascal program of 1.5 kloc designed by two persons.

Publications: [27], [13]

5.3. Pacioli

Participants: Tijs van der Storm, Paul Griffioen [correspondent].

Characterization: A-2-up3, SO-4, SM-2, EM-3, SDL-3-up4, OC-DA-3-CD-3-MS-3-TPM-3.

WWW: http://svn.rascal-mpl.org/pacioli/

Objective: Encapsulate all the variability in the construction of modeling and analysis tools in computational auditing

Users: Financial auditing experts

Impact: Pacioli is an experiment with a big potential in the field of computational auditing. It operates as a vehicle now for experimenting with new ideas in this field. The goal is to tackle the enormous complexity in the (trading) of companies using high level modeling and analysis techniques.

Competition: Pacioli competes with less specialized and less formal business analysis tooling, mostly based on spreadsheets.

Engineering: Pacioli is a part Java, part Rascal project written by one person.

5.4. Rascal

Participants: Paul Klint, Jurgen Vinju [correspondent], Tijs van der Storm, Bas Basten, Jeroen van den Bos, Mark Hills, Bert Lisser, Arnold Lankamp, Atze van der Ploeg, Vadim Zaytsev, Anastasia Izmaylova, Anya Helene Bagge.

Characterization: A5, SO-4, SM-4, EM-4, SDL-4-up5, OC-DA-3-CD-3-MS-3-TPM-3.

WWW: http://www.rascal-mpl.org

Objective: Provide a completely integrated programming language parametric meta programming language for the construction of any kind of meta program for any kind of programming language: analysis, transformation, generation, visualization.

Users: Researchers in model driven engineering, programming languages, software engineering, software analysis, as well as practitioners that need specialized tools.

Impact: Rascal is making the mechanics of meta programming into a non-issue. We can now focus on the interesting details of the particular fact extraction, model, source analysis, domain analysis as opposed to being distracted by the engineering details. Simple things are easy in Rascal and complex things are manageable, due to the integration, the general type system and high-level programming features.

Competition: There is a plethora of meta programming toolboxes and frameworks available, ranging from plain parser generators to fully integrated environments. Rascal is distinguished because it is a programming language rather than a specification formalism and because it completely integrates different technical domains (syntax definition, term rewriting, relational calculus). For simple tools, Rascal competes with scripting languages and for complex tools it competes context-free general parser generators, with query engines based on relational calculus and with term rewriting and strategic programming languages.

Engineering: Rascal is about 100 kLOC of Java code, designed by a core team of three and with a team of around 8 phd students and post-docs contributing to its design, implementation and maintenance. The goal is to work towards more bootstrapping and less Java code as the project continues.

Publications: [21], [28], [29], [22][6], [7]

5.4.1. Novelties

- Re-design of embedded grammar formalism including semantic disambiguation facilities.
- Extremely fast top-down context-free general parsing algorithm in cubic time and space.
- Parse error reporting via partial parse trees (useful in incremental syntax highlighting and incremental type analysis).
- Auto-indent feature for code generation templates.
- Significant extensions and improvements of software visualization library, such as hierarchical graphs and smaller set of more powerful primitives for charts and inter-active features.
- Significant improvements to online documentation and inter-active tutor environment.
- “ToLaTex” mode to include Rascal code in papers.
- ShellExec library for inter-acting via pipes with external programs
- Bridge to Maude and K.
- Generalized function dispatch to arbitrary pattern dispatch.
- New module composition mechanism “extend” next to “import”.
- Ambiguity diagnostics library and parse tree visualizations as a first step towards more grammarware in the IDE.
- A command-line interface to run a single Rascal program.

- Fixed a number of memory leaks in the IDE.
- IDE features for mixed Java/Rascal projects.
- Rational numbers.
- Formal concept analysis library.
- Enhanced SDF2 to Rascal translation.
- Redesigned and simplified abstract grammar format.
- Added “break”, “continue” and “fail” statements for back-tracking and continuation control.
- Radically changed internal design from Visitor to Interpreter design pattern (using an automated refactoring).

5.5. IDE Meta-tooling Platform

Participants: Jurgen Vinju [correspondent], Arnold Lankamp, Anya Helene Bagge.

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.

Characterization: A5, SO-3, SM4-up5, EM-4, SDL-5, DA-2-CD-2-MS-2-TPM-2

WWW: http://www.eclipse.org/imp

Objective: The IDE Meta Tooling Platform (IMP) provides a high-level abstraction over the Eclipse API such that programmers can extend Eclipse with new programming languages or domain specific languages in a few simple steps. IMP also provides a number of standard meta tools such as a parser generator and a domain specific language for formal specifications of configuration parameters.

Users: Designers and implementers of IDEs for programming languages and domain specific languages. Also, designers and implementers of meta programming tools.

Impact: IMP is popular among meta programmers especially for it provides the right level of abstraction.

Competition: IMP competes with other Eclipse plugins for meta programming (such as Model Driven Engineering tools), but its API is more general and more flexible. IMP is a programmers framework rather than a set of generators.

Engineering: IMP is a long-lived project of many contributors, which is managed as an Eclipse incubation project at `eclipse.org`.

Publications: [3]

Jurgen Vinju and Arnold Lankamp contribute significantly to the development of IMP. Their effort is 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.

The Rascal language itself was accepted by Eclipse as a contribution to the IMP project. This will further strengthen the collaboration between the IMP and the Rascal team as well as generate a wider audience for Rascal.

5.6. Ensō

Participant: Tijs van der Storm [correspondent].

Characterization: A5, SO-4, SM-3-up-4, EM-2-up-4, SDL-4, OC-DA-4-CD-4-MS-4-TPM-4

WWW: http://www.enso-lang.org

Objective: Together with Prof. Dr. William R. Cook of the University of Texas at Austin, Tijs van der Storm has been designing and implementing a new programming system, called Ensō. Ensō is theoretically sound and practical reformulation of model-based development. It is based on model-interpretation as opposed to model transformation and code generation. Currently, the system already supports models for schemas (data models), web applications, context-free grammars, diagram editors and security.

Users: All programmers.

Impact: Ensō has the potential to revolutionize the activity of programming. By looking at model driven engineering from a completely fresh perspective, with as key ingredients interpreters and partial evaluation, it may make higher level (domain level) program construction and maintenance as effective as normal programming.

Competition: Ensō competes as a programming paradigm with model driven engineering tools and generic programming and languages that provide syntax macros and language extensions.

Engineering: Ensō is less than 7000 lines of (bootstrapped) Ruby code.

5.7. Software Language Processing Suite

Participant: Vadim Zaytsev [correspondent].

Characterization: A3-up4, SO-4, SM-3, EM-2up3, SDL-2, OC-DA-4-CD-4-MS-4-TPM-4

WWW: http://slps.sourceforge.net

Objective: The project facilitates exposition and comparison of approaches and techniques on language processing.

Users: Computer science students, teachers, engineerings and practitioners

Impact: SLPS contains the largest collection of grammars for programming languages directly recovered from documentation, as well as the largest collection of source-to-source grammar formalisms translators and other related grammarware.

Engineering: SLPS is a large collection of scripts and programs written by Ralf Lämmel and Vadim Zaytsev.

5.7.1. Novelties

- New grammars: Ada, Dart, Eiffel, Fortran, Modula, Mediawiki, ...(now a total of 41)
- Grammar Tank: a new collection of 54 small grammars for research purposes
- TestMatch: a tool for grammar-based differential testing of ANTLR grammars and for nonterminal matching based of parsing generated test data (in collaboration with Ralf Lämmel).
- Grammar Hunter: a tool for automated notation-parametric grammar recovery (will also be a Rascal library).

5.8. Demo Light for Composing Models

Participants: Jan van Eijck [correspondent], Floor Sietsma.

Characterization: A2,SO-3,SM-1,EM-2,SDL-2,OC-4

WWW: http://homepages.cwi.nl/~jve/software/demolight0/

Objective: Demonstrate epistemic modeling and reasoning

Users: Students and researchers in application of epistemic logic

Impact: Demo light makes the theory of epistemic reasoning insightful by offering a Haskell library for experimenting with it.

Engineering: Demo Light is a Haskell library.

6. New Results

6.1. Ambiguity Detection in Context-free Grammars

The work on static detection of ambiguity in context-free grammars continued in 2011. Bas Basten has worked on scaling previous results to so-called character level grammars. This includes the application of declarative disambiguation filters that increase the efficiency as well as the accuracy of the analysis [14].

6.2. Automated Diagnostics for Ambiguity in Context-free Grammars

When an ambiguity is found this is reported by a complex trace (usually a set of parse trees). It is difficult for a human to spot the cause of the ambiguity and devise a fitting solution. The Dr Ambiguity algorithm is an expert tool, by Bas Basten and Jurgen Vinju, that compares different parse trees for the same sentence on essential attributes that can be distinguished by declarative disambiguation methods [15].

6.3. A general library for software visualization

The Rascal standard library was extended with a very flexible component for the rapid construction of new/experimental (inter-active) visualizations. The current library is fully working and forms the inspiration for a possible domain-specific extension of the Rascal meta programming language to be integrated at a later stage [23]. This is work by Paul Klint and Atze van der Ploeg.

6.4. Comparing Design Patterns - the case of Visitor versus Interpreter

In this research application of Rascal, Mark Hills, Tijs van der Storm, Paul Klint and Jurgen Vinju focused on analyzing the emergent differences in quality when choosing between two different source code design patterns. We constructed a refactoring tool that translates instances of the Visitor design pattern to the Interpreter design pattern in a semi-automated fashion. This then allowed us to study two versions of an otherwise equivalent system in terms of efficiency and maintainability [24].

6.5. Entry in the language workbench competition

Tijs van der Storm and Jurgen Vinju participated on the Language Workbench Competition 2011 (LWC'11), showcasing the DSL construction and capabilities of Rascal. The objective was the development of a number of DSLs for entity-relation modeling. The modular implementation of these languages was documented in a technical report [35] The DSL implementation featured modular context-free grammars for parsing, modular type checkers, modular code generators and full-fledged IDEs (syntax highlighting, outlining, error marking, etc.). The complete implementation only takes around 700 lines of Rascal code.

6.6. A compiler for OBERON-0

Tijs van der Storm led the participation of ATEAMS in the LDTA Tool Challenge 2011. This was a collaborative effort together with Atze van der Ploeg, Mark Hills, Bas Basten, Paul Klint, Bert Lisser, Jeroen van den Bos, Jurgen Vinju and Arnold Lankamp. The objective of the challenge was to implement all aspects of a simple, imperative language called Oberon-0. The components that had to be implemented included: parsing, pretty printing, name analysis, type checking, and compilation to C. Additionally, the components should be developed in a modular fashion according to four language levels: each language level added more language features and required the modular extension of the components. Although not required for the challenge, we also developed a compiler targeting Java, a compiler targeting JVM byte code, a control-flow graph visualizer, and IDE support. The implementation required only around 4200 lines of Rascal code. The result was presented at the international workshop on Language Descriptions Tools and Applications.

6.7. Generalized Grammar Recovery

Vadim Zaytsev managed to generalize the algorithm for recovering context-free grammars from legacy language documentation. This facilitates the recovery of a lot more grammars to be used in the study of grammarware and software language engineering.

6.8. Comparing Context-free Grammars through test data

Equivalence of the languages generated by two different context-free grammars is undecidable, yet language equivalence is one of the most important quality aspects of context-free grammars for programming languages. Vadim Zaytsev introduced and experimentally validated a method based on differential analysis: generating sentences from both grammars and cross-testing them.

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.
- ATEAMS was contracted by Open Universiteit for the course material and support in teaching the Software Evolution course.

8. Partnerships and Cooperations

8.1. National Initiatives

+ NWO TOP proposal “Domain-Specific Languages: A Big Future for Small Programs”
+ “Next Generation Auditing: Data Assurance as a Service” (Jacquard project)
+ “Escher: End-user SCripting for High-level softwarE Representation” (NWO)
+ “GrammarLab: Foundations For a Grammar Laboratory” (NWO)
+ “Model Driven Engineering in Digital Forensics” (NWO)
+ “EQuA: Early Quality Assurance in Software Production”

8.2. International Initiatives

8.2.1. Visits of International Scientists

- Prof. Dr. Michael Godfrey (University of Waterloo, Canada) is visiting for one year, starting August 2011.
- Dr. Anya Helene Bagge (Institutt for Informatikk Universitetet i Bergen, Norway) was visiting us for one year until October 2011.
- Prof. Dr. William Cook visited us for a week (University of Texas, U.S.A.)
- Prof. Dr. Eric van Wyk visited us for one week (University of Minnesota, U.S.A.)
- Prof. Dr. Terence Parr visited us for two days (University of San Francisco, U.S.A.)
- Dr. Markus Völter visited us for two days (Germany)
- Dr. Wolgang Lohmann visited us for a day (EMPA, Switserland)
- Prof. Dr. Ralf Lämmel visited us for a week (Universität Koblenz-Landau, Germany)
- Dr. Vlad Rusu visited us several times for two days (INRIA Lille, France)

8.2.1.1. Internships

- Wietse Venema 2011/2012
- Randy Fluit 2011
- Ahmadi Nasab 2010/2011
- Jouke Stoel 2011
- Christian Köppe 2011
- Davy Landman 2011
- Jan de Mooij 2011

9. Dissemination

9.1. Animation of the scientific community

- Jan van Eijck : Member of the European Network in Computational Logic
- Jan van Eijck : Member of the International Quality Assessment Committee
- Mark Hills : Participation WG 2.11 Program Generation
- Mark Hills : Program committee chair K workshop
- Mark Hills : Reviewing Journal of Logic and Algebraic Programming
- Mark Hills : Reviewing Science of Computer Programming
- Mark Hills : Reviewing Software: Practice and Experience
- Mark Hills : Keynote lecture (AMMSE) for Computer Science and AI in Flanders, Belgium
- Paul Klint : Editor for Science of Computer Programming
- Paul Klint : Editor for Springer Service Science Book Series
- Paul Klint : Visiting Professor University of London, Royal Holloway
- Paul Klint : Head of Software engineering Department, CWI
- Paul Klint : Head of CWI Research group Software Analysis and Transformation (SEN1)
- Paul Klint : Full Professor at UvA, Software Engineering Chair

- Paul Klint : Director Master Software Engineering, UvA
- Paul Klint : Treasurer European Association for Programming Languages and Systems (EAPLS)
- Paul Klint : Steering committee member ETAPS
- Paul Klint : Board member Instituut voor Programmatuur en Architectuur (IPA)
- Paul Klint : External advisor PlanComps project (UK)
- Paul Klint : Program committee CEE-SET
- Paul Klint : Program committee EVOL
- Paul Klint : Program committee ICECCS
- Paul Klint : Program committee SERENE
- Paul Klint : Program committee WasDETT
- Paul Klint : Program committee SLE
- Paul Klint : Program committee SLE
- Paul Klint : Co-organizer INRIA Software Engineering Conference (Amsterdam)
- Paul Klint : Keynote speaker SLE
- Paul Klint : Invited speaker PlanCoMPS meeting (Swansea)
- Paul Klint : Invited speaker XBRL (Kansas, U.S.A.)
- Paul Klint : Invited speaker AIESEC (Utrecht, The Netherlands)
- Paul Klint : PhD committee Jeroen Arnoldus, Technical University Eindhoven, Feb 1, 2011
- Paul Klint : PhD committee Thomas Bernard, University of Amsterdam, March 11, 2011
- Paul Klint : PhD committee Damien Cassou, University of Bordeaux, March 17, 2011
- Paul Klint : PhD committee Lennart Kats, Delft University, December 13, 2011
- Tijs van der Storm : Participation WG 2.11 Program Generation
- Tijs van der Storm : Invited speaker Scientific Meeting CWI
- Jurgen Vinju : 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)
- Jurgen Vinju : Program committee ESEC/FSE
- Jurgen Vinju : Program committee GPCE
- Jurgen Vinju : Program committee ICMT
- Jurgen Vinju : Program committee K Workshop
- Jurgen Vinju : Program committee LOPSTR
- Jurgen Vinju : Program committee SCAM
- Jurgen Vinju : Steering committee SCAM
- Jurgen Vinju : Steering committee SLE
- Jurgen Vinju : Program committee SLE
- Jurgen Vinju : Workshop selection chair SLE
- Jurgen Vinju : Program committee SQM
- Jurgen Vinju : Program committee WASDETT
- Jurgen Vinju : Invited speaker Theoretical Computer Science Amsterdam
- Jurgen Vinju : Secretary Van Wijngaarden Awards CWI

- Jurgen Vinju, Mark Hills, Tijs van der Storm : Participation WG 2.11 Program Generation
- Jurgen Vinju : Reviewing Journal of Software Maintenance and Evolution: Research and Practice
- Jurgen Vinju : Reviewing Transactions on Software Engineering (TSE)
- Vadim Zaytsev : Publicity chair GTTSE
- Vadim Zaytsev : Publicity co-chair SLE
- Vadim Zaytsev : Program chair of Wikimedia Conference Netherlands (WCN)
- Vadim Zaytsev : Program committee SCAM
- Vadim Zaytsev : Program committee SCAM
- Vadim Zaytsev : Invited talk TU Delft
- Vadim Zaytsev : Wikimania presentation “Wiki migration using Rascal”

9.2. Teaching

ATEAMS taught the following courses in 2011:

Master: Software Evolution, 8 hours per week, Universiteit van Amsterdam, The Netherlands. Jurgen Vinju responsible, assistants Paul Klint, Vadim Zaytsev, Tijs van der Storm, Anastasia Izmaylova, Davy Landman and Atze van der Ploeg.

Master: Software Construction, 8 hours per week, Universiteit van Amsterdam, The Netherlands. Tijs van der Storm responsible, assistants Jurgen Vinju and Mark Hills.

Master: Software Testing, 8 hours per week, Universiteit van Amsterdam, The Netherlands. Jan van Eijck responsible, assistants Bert Lisser and Floor Sietsma.

Bachelor: 5 language Summer school (Rascal day), 8 hours, Tijs van der Storm responsible, assistants Mark Hills, Jeroen van den Bos and Atze van der Ploeg.

Master: Software Evolution, Open Universiteit, The Netherlands. Paul Klint and Jurgen Vinju facilitate two guest courses and the course material.

These are the PhD students of ATEAMS in 2011:

PhD : Bas Basten, Ambiguity Detection for Programming Language Grammars, Universiteit van Amsterdam, December 15th 2011, supervisors Paul Klint, Jurgen Vinju

PhD in progress : Paul Griffioen, Next Generation Computational Auditing, started 2011, supervisors Paul Klint, Philip Elsas

PhD in progress : Anastasia Izmaylova, A General Language Parametric Framework for Software Refactoring, started 2011, supervisors Paul Klint, Jurgen Vinju

PhD in progress : Jeroen van den Bos, Digital Forensics Software Engineering, started 2010, supervisors Paul Klint, Tijs van der Storm

PhD in progress : Atze van der Ploeg, Rapid Language Parametric Prototyping of Software Visualization and Exploration, started 2011

PhD in progress : Davy Landman, Recovery and Synthesis of Domain Specific Language Design, started 2011

PhD in progress : Floor Sietsma, Logics of Communication and Knowledge, started 2009, supervisors Jan van Eijck, Krzysztof Apt

PhD in progress : Riemer van Rozen, Software Engineering Principles for the Gaming Domain, started 2011, supervisor Paul Klint

ATEAMS members supervised the following masters theses in 2011:

Christian Köppe, DoKRe - A Method for Automated Domain Knowledge Recovery from Source Code

Jouke Stoel, Finding Naming Bugs with Formal Concept Analysis (to appear)

Davy Landman, Exploring Methods for Locating Features in Software Systems : a DSL Perspective

Gijs van Lammeren, Automatic Scaling of the Windows Azure Platform : A study on the efficiency of scaling policies

Jan de Mooij, Testcase Generation based on Property-based Testing

Joost Pastoor, Sensor query performance optimization by caching

Lennart Tange, Test-based modeling

Roberto van der Horst, The Influence of First-Class Relations on Coupling and Cohesion : A Case Study

Marvin Jacobsz, Een performance analyse van “Hiphop for PHP”

Aart van den Dolder, Bepaling van de geschiktheid van Oracle Forms applicaties voor inbeheername door middel van automatische code review volgens het SIG Maintainability model

Randy Fluit, Differencing Context-free Grammars

Willem Fibbe, Calculating detailed source code metrics and applying them to find code clones

Freddy Offenga, Aanleiding tot refactoring : Een exploratief onderzoek naar detectie en uitvoering van de Rename Method refactoring

Rob Smit, Idioms-based Business Rule Extraction

Lammert Vinke, Estimate the post-release Defect Density based on the Test Level Quality

Lennart Tange, Test-based Modeling

Niels Beekman, Evaluation of storage mechanisms for a content repository

Maarten Hoekstra, Static source code analysis with respect to ORM performance anti-patterns

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] 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.

[4] 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.

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

[6] 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.

[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. LÄMMEL, J. SARAIVA, J. VISSER (editors), Lecture Notes in Computer Science, Springer, Heidelberg, 2010, n^{o} 6491, p. 222–298.

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

Publications of the year

Doctoral Dissertations and Habilitation Theses

[9] B. BASTEN. *Ambiguity Detection for Programming Language Grammars*, Universiteit van Amsterdam, December 2011, http://hal.inria.fr/tel-00644079/en.

Articles in International Peer-Reviewed Journal

[10] R. LÄMMEL, V. ZAYTSEV. *Recovering Grammar Relationships for the Java Language Specification*, in "Software Quality Journal", 2011, vol. 19, n^{o} 2, p. 333–378 [*DOI :* 10.1007/s11219-010-9116-5], http://hal.inria.fr/hal-00645313/en.

[11] E. PACUIT, S. SIMON. *Reasoning with Protocols under Imperfect Information*, in "The Review of Symbolic Logic", 2011, vol. 4, n^{o} 3, p. 412-444, http://hal.inria.fr/hal-00644259/en.

Articles in National Peer-Reviewed Journal

[12] J. V. EIJCK. *Redeneren over Communicatie*, in "Euclides", 2011, vol. 86, n^{o} 11, http://hal.inria.fr/hal-00645297/en.

International Conferences with Proceedings

[13] L. ARONSON, J. VAN DEN BOS. *Towards an Engineering Approach to File Carver Construction*, in "2011 IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW)", Munchen, Germany, IEEE, 2011, p. 368-373, http://hal.inria.fr/hal-00644688/en.

[14] B. BASTEN, P. KLINT, J. VINJU. *Ambiguity Detection : Scaling to Scannerless*, in "Proceedings of the Fourth International Conference on Software Language Engineering (SLE 2011)", Braga, Portugal, J. SARAIVA, U. ASSMANN, A. SLOANE (editors), Springer, July 2011, vol. 6940, http://hal.inria.fr/hal-00644053/en.

[15] B. BASTEN, J. VINJU. *Parse Forest Diagnostics with Dr. Ambiguity*, in "Proceedings of the Fourth International Conference on Software Language Engineering (SLE 2011)", Braga, Portugal, J. SARAIVA, U. ASSMANN, A. SLOANE (editors), 2011, vol. 6940, http://hal.inria.fr/hal-00644063/en.

[16] J. v. EIJCK. *A Geometric Look at Manipulation*, in "CLIMA XII 2011", Barcelona, Spain, J. LEITE (editor), Springer, 2011, vol. 6814, p. 92–104, http://hal.inria.fr/hal-00645301/en.

[17] J. v. EIJCK, F. SIETSMA. *Message Passing in a Dynamic Epistemic Logic Setting*, in "TARK XIII: Proceedings of hte 13th Conference on Theoretical Aspects of Rationality and Knowledge", New York, United States, K. APT (editor), ACM, 2011, p. 212–220, http://hal.inria.fr/hal-00645304/en.

[18] J. v. EIJCK, F. SIETSMA. *Message-Generated Kripke Semantics*, in "Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)", Taipei, Taiwan, Province Of China, P. YOLUM, K. TUMER, P. STONE, L. SONENBERG (editors), 2011, p. 1183–1184, http://hal.inria.fr/hal-00645298/en.

[19] J. v. EIJCK, F. SIETSMA, S. SIMON. *Reflections on Vote Manipulation*, in "Proceedings of LORI 2011", Guangzhou, China, H. P. VAN DITMARSCH, J. LANG, S. JU (editors), Springer, 2011, vol. 6953, p. 386-387, http://hal.inria.fr/hal-00644260/en.

[20] B. FISCHER, R. LÄMMEL, V. ZAYTSEV. *Comparison of Context-free Grammars Based on Parsing Generated Test Data*, in "Post-proceedings of the Fourth International Conference on Software Language Engineering (SLE 2011)", Braga, Portugal, U. ASSMANN, A. SLOANE (editors), Springer, Heidelberg, 2011, vol. 6940, http://hal.inria.fr/hal-00645306/en.

[21] P. KLINT, M. HILLS, J. VAN DEN BOS, T. VAN DER STORM, J. VINJU. *Rascal: From Algebraic Specification to Meta-Programming*, in "Proceedings Second International Workshop on Algebraic Methods in Model-based Software Engineering (AMMSE)", Zurich, Switzerland, 2011, p. 15-32, http://hal.inria.fr/hal-00644689/en.

[22] P. KLINT, J. VINJU, M. HILLS. *RLSRunner: Linking Rascal with K for Program Analysis*, in "Proceedings of the Fourth International Conference on Software Language Engineering (SLE 2011)", Braga, Portugal, Springer, 2011, vol. 6940, http://hal.inria.fr/hal-00644695/en.

[23] B. LISSER, P. KLINT, A. VAN DER PLOEG. *Towards a One-Stop-Shop for Analysis, Trasnformation anf Visualization of Software, Proceedings Solftware Language Engineering*, in "Fourth International Conference on Software Language Engineering", Braga, Portugal, July 2011, http://hal.inria.fr/hal-00645323/en.

[24] J. VINJU, P. KLINT, M. HILLS, T. VAN DER STORM. *A Case of Visitor versus Interpreter Pattern*, in "Proceedings of the 49th International Conference on Objects, Models, Components and Patterns", Zurich, Switzerland, 2011, http://hal.inria.fr/hal-00644685/en.

[25] V. ZAYTSEV, R. LÄMMEL. *A Unified Format for Language Documents*, in "Post-proceedings of the Third International Conference on Software Language Engineering (SLE 2010)", Braga, Portugal, B. MALLOY, S. STAAB, M. G. J. VAN DEN BRAND (editors), Springer-Verlag, 2011, vol. 6563, p. 206–225 [*DOI :* 10.1007/978-3-642-19440-5_13], http://hal.inria.fr/hal-00645324/en.

[26] V. ZAYTSEV. *Language Convergence Infrastructure*, in "Post-proceedings of the Third International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE 2009)", Braga, Portugal, J. MIGUEL, R. LÄMMEL, J. VISSER, J. SARAIVA (editors), Springer-Verlag, 2011, vol. 6491, p. 481–497 [*DOI :* 10.1007/978-3-642-18023-1_16], http://hal.inria.fr/hal-00645316/en.

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

National Conferences with Proceeding

[28] J. VAN DEN BOS, P. GRIFFIOEN, T. VAN DER STORM. *Rascal: Language Technology for Model-Driven Engineering*, in "ICT.Open", Veldhoven, Netherlands, 2011, http://hal.inria.fr/hal-00644690/en.

[29] J. VINJU, M. HILLS, P. KLINT, A. VAN DER PLOEG, A. IZMAYLOVA, T. VAN DER STORM. *The Rascal meta-programming language – a lab for software analysis, transformation, generation & visualization*, in "ICT.Open", Veldhoven, Netherlands, November 2011, http://hal.inria.fr/hal-00644693/en.

Scientific Books (or Scientific Book chapters)

[30] J. V. BENTHEM, J. VAN EIJCK, J. JASPARS, H. VAN DITMARSCH. *Logic in Action*, Internet, 2011, http://hal.inria.fr/hal-00646013/en.

[31] N. DIMITRI, J. V. EIJCK. *Time discounting and time consistency*, in "Games, Actions, and Social Software", J. V. EIJCK, R. VERBRUGGE (editors), Springer Verlag, 2011, vol. 7010, p. 31–41, http://hal.inria.fr/hal-00645290/en.

[32] J. V. EIJCK. *Perception and Change in Update Logic*, in "Games, Actions, and Social Software", J. V. EIJCK, R. VERBRUGGE (editors), Springer Verlag, 2011, vol. 7010, p. 129–151, http://hal.inria.fr/hal-00645291/en.

[33] J. V. EIJCK, H. KAMP. *Discourse Representation in Context*, in "Handbook of Logic and Language", J. V. BENTHEM, A. TER MEULEN (editors), Elsevier, 2011, http://hal.inria.fr/hal-00645296/en.

Research Reports

[34] J. VINJU. *SDF Disambiguation Medkit for Programming Languages*, INRIA CWI, 2011, http://hal.inria.fr/hal-00644091/en.

[35] T. VAN DER STORM. *The Rascal Language Workbench*, INRIA CWI, 2011, http://hal.inria.fr/hal-00645985/en.

Other Publications

[36] V. ZAYTSEV. *MediaWiki Grammar Recovery*, 2011, Computing Research Repository, http://hal.inria.fr/hal-00645308/en.