IN PARTNERSHIP WITH:
Centrum Wiskunde & Informatica

Activity Report 2013

Project-Team ATEAMS

Analysis and Transformation based on rEliAble tool coMpositionS

RESEARCH CENTER
Lille - Nord Europe

THEME
Architecture, Languages and Compilation

Table of contents

Project-Team ATEAMS

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

Creation of the Project-Team: 2009 July 01.

1. Members

Research Scientists
Jurgen Vinju [Team leader, CWI]
Paul Klint [CWI, Professor]
Tijs Van Der Storm [CWI]
Jan Van Eijck [CWI, Professor]
Robert Van Liere [CWI, Professor]

Engineers
Maarten Dijkema [CWI]
Bert Lisser [CWI]

PhD Students
Ali Afroozeh [CWI]
Paul Griffioen [CWI]
Pablo Inostroza Valdera [CWI]
Anastasia Izmaylova [CWI]
Davy Landman [CWI]
Michael Steindorfer [CWI]
Jeroen Van Den Bos [CWI, until Aug 2013]
Atze Van Der Ploeg [CWI]
Riemer Van Rozen [CWI]
Ashim Shahi [CWI]

Post-Doctoral Fellows
Mark Hills [CWI, until Sep 2013]
Floor Sietsma [CWI, until Dec 2013]
Sunil Simon [CWI, until Aug 2013]
Vadim Zaytsev [CWI, until Nov 2013]

Administrative Assistant
Sandrine Meilen [Inria]

2. Overall Objectives

2.1. Presentation

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 Each system is large, uses incompatible — new, outdated or non-standard — technology and above all, exhibits failures.

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

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

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

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

Finally, we study code generation from domain specific languages (DSLs). Here we also find a combination of analysis and transformation. Designing, implementing and, very importantly, maintaining DSLs is costly. We focus on application areas such as Computational Auditing and Digital Forensics to experiment with this subject. In Computational Auditing we are focusing on modeling interactive questionnaires and in Digital Forensics on optimization and specialization of file format recognition.

2.2. Highlights of the Year

- Paul Klint was Knighted Officer in the order of Oranje Nassau based on his contributions to science and education.
- Paul Klint was appointed Research Fellow, Centrum Wiskunde & Informatica

3. Research Program

3.1. Research method

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

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

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

3.2. Software analysis

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

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

The approach we are exploring is the use of syntax-related program patterns for fact extraction. An early proposal for such a pattern-based approach 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 closure) and the performance problems encountered, this approach has not seen wider use.

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

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

3.2.1. Goals

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

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

3.3. Refactoring and Transformation

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

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

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

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

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

3.3.1. Goals

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

3.4. The Rascal Meta-programming language

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

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

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

3.4.1. Goals

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

3.5. Domain-specific Languages

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

Language workbenches [24] are integrated environments to facilitate the development of all aspects of DSLs. This includes IDE support (e.g., syntax coloring, outlining, reference resolving etc.) for the defined languages. Rascal can be seen as a language workbench that focuses on flexibility, programmability and modularity. DSL implementation is, in essence, an instance of source code analysis and transformation. As a result, Rascal's features for fact extraction, analysis, tree traversal and synthesis are an excellent fit for this area. An important aspect in this line of research is bringing the IDE closer to the source code. This will involve investigation of heterogeneous representations of source code, by integrating graphical, tabular or forms-based user interface elements. As a result, we propose Rascal as a feature-rich workbench for model-driven software development.

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

3.5.1. Goals

The goal of this research topic is to improve the practice of DSL-based software development through language design and tool support. A primary focus is to extend the IDE support provided by Rascal, and to facilitate incremental, and iterative design of DSLs. The latter is supported by new (meta-)language constructs for extending existing language implementations. This will require research into extensible programming and composition of compilers, interpeters and type checkers. Finally, a DSL is never an island: it will have to integrate with (third-party) source code, such as host language, libraries, runtime systems etc. This leads to the vision of multi-lingual programming environments [22]. .

4. Software and Platforms

4.1. MicroMachinations

Participant: Riemer Van Rozen [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:

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

Users: Game designers

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

Competition: None.

Engineering: The front-end of MicroMachinations is built using the Rascal language workbench, including visualization, model checking, debugging and standard IDE features. The runtime library is implemented in C++ and will be evaluated in the context of industrial game design.

Publications: [28]

4.1.1. Novelties

- Development on MMLib was started which allows the execution of game economies directly within games.

4.2. 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://www.derric-lang.org

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: [35], [34][14], [16], [15]

4.2.1. Novelties

- Construction of a 1TB benchmark based on Wikipedia images.
- The Derric DSL for digital forensics now features Trinity, a runtime IDE to debug file format descriptions [35].

4.3. Rascal

Participants: Paul Klint, Jurgen Vinju [correspondent], Tijs Van Der Storm, Jeroen Van Den Bos, Mark Hills, Bert Lisser, Atze Van Der Ploeg, Vadim Zaytsev, Anastasia Izmaylova, Michael Steindorfer, Ali Afroozeh, Ashim Shahi.

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: [7], [6], [8], [5], [6]

4.3.1. Novelties

- A new language-parametric model to represent software projects, called M3 [38].
- Performance improvements of the Rascal interpreter throughout.
- Initial version of a compiler for Rascal, based on new language construct guarded coroutines.
- Origin tracking for values and expressions of type string.
- A library for accessing and analyzing Excel and Word documents.
- Improvements to the Rascal IDE: better output handling, hyper linked source code locations in the console, dedicated project explorer view.
- Content completion for DSLs implemented in Rascal.
- Significant improvements to the Rascal static type checker.
- Experiments with improved GLL parsing (Iguana).
- Several new example DSL implementations to illustrate Rascal as a language workbench: Marvol, a DSL for controlling NAO robots, and two implementations of a DSL for questionnaires (DemoQLes and QL-R-Kemi).

4.4. IDE Meta-tooling Platform

Participants: Jurgen Vinju [correspondent], Michael Steindorfer.

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: https://github.com/impulse-org/

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`. Currently we are moving the project to Github to explore more different ways of collaboration.

Publications: [2]

4.4.1. Novelties

- The IMP program database (PDB) was completely redesigned.

4.5. 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, and Alex Loh, 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 a completely self-hosted system in 7000 lines of code.

Publications: [12], [17], [11]

4.5.1. Novelties

- A compiler for a dedicated Ensō language, which targets JavaScript.
- Added a demo based on the LWC'13 questionnaire language assignment.

4.6. LiveQL

Participant: Tijs Van Der Storm [correspondent].

Characterization: A1, SO-3, SM-1, EM-2, SDL-4, OC-DA-4-CD-4-MS-4-TPM-4

WWW: https://github.com/cwi-swat/liveql

Objective: Experimenting with live programming concepts and techniques in the context of domain specific languages (DSLs).

Users: End-user programmers.

Impact: LiveQL is an experiment in making a DSL "live", i.e. any change to the DSL program is immediately reflected in the running program. This has the potential to widen the audience of DSL users to include end-user programmers.

Competition: The end-goal is to provide live end-user programming environments with domain-specific checking and optimization. The most similar tools are spreadsheet applications. However, these are still quite general.

Engineering: LiveQL is built in Java, using the ANTLR parser generator.

Publications: [36]

4.7. QL-R-Kemi

Participant: Tijs Van Der Storm [correspondent].

Characterization: A1, SO-3, SM-1, EM-2, SDL-4, OC-DA-4-CD-4-MS-4-TPM-4

WWW: https://github.com/cwi-swat/QL-R-Kemi

Objective: Demonstrate the language workbench features of the Rascal meta programming language and environment. Investigate domain specific language application in the domain of questionnaires and surveys.

Users: Students, scientists.

Impact: Questionnaires are common in social science, tax administration and statistics. Discovering the right abstractions for describing questionnaires has the potential to significantly improve the practice of constructing questionnaire software.

Competition: Traditional survey tools are often wizard-based, lack computational capabilities and lack a formal foundation. The same language is built in a number of different language workbenches which served as a benchmark to compare such tools [24].

Engineering: Uses all features of the Rascal language workbench.

Publications: [24]

5. New Results

5.1. Empirical analyses of source code

Rascal was used to perform empirical investigations of existing source code bases. First of all, Davy Landman performed an analysis of project management source code to investigate if domain knowledge is present in source code and, if so, how easy it is to extract that knowledge [26]. An earlier experiment in static analysis of PHP code was finalized by Mark Hills. The result is a deep study of feature usage in a large number of well-known PHP projects [25]. Vadim Zaytsev conducted an experiment to recognize micro-patterns in grammars and meta-models [32]. Finally, Jeroen van den Bos performed a deep empirical study to find out as to how far a domain-specific language facilitates evolution [34]. The results showed that the Derric DSL did indeed cover most evolution scenarios, but there is still room for improving the language. In all cases Rascal proved to be instrumental in performing the experiments.

5.2. Better parsing and disambiguation

Ali Afroozeh worked on a new implementation of GLL parsing, called Iguana. Unlike traditional parser generators, Iguana adopted the interpretive approach that is also used in the Ensō parser. This experiment is still ongoing, but the new parser is expected to be integrated into Rascal beginning of 2014. Additionally, a longstanding problem of disambiguation using operator precedence was solved [23]. Traditional approaches are either not safe (i.e. they make the language smaller), or they do not support complex precedence rules as found in, for instance, OCaml.

5.3. Extensible Programming

Modular and extensible implementation of languages could have major impact on how DSLs will be implemented. Anastasia Izmaylova continued here work on improving the extensibility of Rascal's module system, by providing open recursive function combinators.

Extensible programming is traditionally plagued by what has become known as “the expression problem”, which captures the fact that most programming languages either support extension of data variants, or extension of operations, but not both. Object Algebras are simple solution to this problem. In [30] we have extended this model to support feature-oriented programming. These results are currently being integrated into the Ensō system.

5.4. DSLs for Games

In collaboration with the Hogeschool van Amsterdam, Riemer van Rozen developed a workbench for MicroMachinations, a DSL for game economies [28]. Completely built using Rascal, this DSL environment features syntax highlighting, static analysis, interactive simulation, and SPIN-based model-checking of process models describing the economy of a game. The project shows the versatility of Rascal as a language workbench for the development of DSLs.

5.5. DSLs for Questionnaires

In the context of computational auditing we have intensified our research on DSLs for questionnaires. It was proposed by Tijs van der Storm as the benchmark task for the Language Workbench Challenge 2013 (LWC'13), which has resulted in a thorough overview and qualitative comparison of language workbenches [24]. As a side-effect, there are now two publicly available Rascal implementations of the questionnaire DSL (QL-R-Kemi and Demoqles). A first step has been made to collect all implementations to create a “chrestomathy” for further study and dissemination of language workbench concepts and DSL implementation patterns. Other results include a formal semantics of the dynamics of questionnaires [21], and an initial prototype of a questionnaire model for modeling the Dutch Tax Income filing application by Pablo Inostroza Valdera.

5.6. Live Programming

Live programming aims to bring the dynamic execution of programs closer to the programmer, ideally almost obliterating the gap between editing and executing the program. We are working on applying such principles in the context of DSLs. This has lead to two results: a live programming environment for a DSL for questionnaires [36], and Trinity, a data-driven IDE for Derric [35]. Riemer van Rozen has worked on applying similar techniques to MicroMachinations, so that game economies can be adapted at runtime.

5.7. Visualization and interaction

Atze van der Ploeg worked on designing new algorithms and abstractions in the domain of visualization and abstraction. His first result is a fast algorithm for drawing non-layered, tidy trees [20]. DeForm is a library for the declarative specification of resolution-independent 2D graphics [27]. In [31] he proposed a reformulation of the traditional functional reactive programming (FRP) framework, which is both simple and efficient to implement.

5.8. Guarded Coroutines

Anastasia Izmaylova and Paul Klint have built an initial version of a compiler for Rascal. The performance improvements with respect to the interpreter are impressive. Moreover, the design of compiler is based on a new construct for implementing languages with complex backtracking and pattern matching semantics: guarded coroutines. This construct will be instrumental in extending the Rascal language with new kinds of control-flow and concurrency.

5.9. Data structures for meta programming

The efficiency of many meta programs is dependent on the internal data structures used to represent collections, trees, relations etc. Michael Steindorfer has worked on comparing the performance of various persistent collection libraries (e.g., those used in Rascal, Clojure, and Scala). This has lead to a redesign of the PDB collection library that underlies the data structures of Rascal. Furthermore, he developed the Orpheus tool, an object redunancy profiler to assess the effects of maximal sharing.

6. Partnerships and Cooperations

6.1. National Initiatives

6.1.1. Master Software Engineering

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

6.1.2. Early Quality Assurance in Software Production

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

6.1.3. Model-Driven Engineering in Digital Forensics

In this project ATEAMS works with the Dutch National Forensics Institute on next generation carving software for recovering evidence from damaged or erased data storage media.

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

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

6.2. European Initiatives

6.2.1. FP7 Projects

OSSMETER aims to extend the state-of-the-art in the field of automated analysis and measurement of open-source software (OSS), and develop a platform that will support decision makers in the process of discovering, comparing, assessing and monitoring the health, quality, impact and activity of open-source software. The project started in October 2012. ATEAMS contributes to this project by focusing on software analysis and related areas.

6.3. International Research Visitors

6.3.1. Visits of International Scientists

- Oscar Nierstrasz, PhD, Professor - Professor of Computer Science at the Institute of Computer Science (IAM) of the University of Bern
- Anya Helene Bagge, PhD - University of Bergen, Norway
- Sebastian Erdweg, PhD - TU Darmstadt

6.3.1.1. Internships

- Kevin van der Vlist
- Davy Meers
- Wouter Kwakernaak
- Jimi van der Woning
- Ioana Rucareanu
- Ioannis Tzanellis
- George Marminidis
- Vlad Lep
- Dimitrios Kyritsis
- Chris Mulder

7. Dissemination

7.1. Scientific Animation

- Jan van Eijck : Member of the NWO committee “Vrije Competitie” for Computer Science.
- Jan van Eijck : Member of the Advisory Board (‘Raad van Advies’) of the Artificial Intelligence Curriculum, University of Groningen (since Summer 2013).
- Jan van Eijck : Program committee Tenth International Conference on Computational Semantics (IWCS) University of Potsdam, Germany, March 2013
- Jan van Eijck : Program committee LORI-4 (4th International Workshop on Logic, Rationality and Interaction),
- Jan van Eijck : Program committee TTNLS-2014 (Type Theory for Natural Language Semantics)
- Jan van Eijck : Editor of Journal of Logics and their Applications (new IfCoLog journal with open access, to be published by College Publications).
- Jan van Eijck : Reviewer Artificial Intelligence
- Jan van Eijck : Reviewer ESSLLI
- Jan van Eijck : Reviewer Journal of Semantics,
- Jan van Eijck : Reviewer Journal of Logic and Computation
- Jan van Eijck : Reviewer Fundamenta Informaticae
- Jan van Eijck : Reviewer Synthese,
- Jan van Eijck : Reviewer Journal of Philosophical Logic
- Jan van Eijck : Reviewer Journal of Logic Language and Information
- Jan van Eijck : Reviewer Cambridge University Press
- Jan van Eijck : Reviewer Studia Logica.

- Mark Hills : Program committee Working Conference on Reverse Engineering (WCRE/CSMR) Tool-track
- Mark Hills : Reviewer International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)
- Mark Hills : Program committee CALCO Tools
- Paul Klint : Editor Science of Computer Programming
- Paul Klint : Editor Springer Service Science Book Series
- Paul Klint : Visiting Professor University of London, Royal Holloway
- 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 : Full Professor at UvA, Software Engineering Chair
- Paul Klint : Director Master Software Engineering, UvA
- Paul Klint : Program committee Software Language Engineering (SLE)
- Paul Klint : Program committee Scalable Language Specifications (SLS 2013)
- Paul Klint : Program committee WasDETT 2013
- Paul Klint : Program committee CSMR WCRE ERA 2014
- Paul Klint : EAPLS PhD Awards 2013
- Atze van der Ploeg : Reviewer Journal of Universal Computer Science
- Tijs van der Storm : Program committee International Workshop on Advanced Software Development Tools and Techniques - (WASDeTT)
- Tijs van der Storm : Program committee International Conference on Generative Programming: Concepts & Experiences (GPCE)
- Tijs van der Storm : Program committee Working Conference on Reverse Engineering (WCRE/CSMR) Tool-track
- Tijs van der Storm : Reviewer ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)
- Tijs van der Storm : Reviewer ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)
- Tijs van der Storm : Member working group IFIP TC2 WGLD 2.16: Working Group on Language Design
- Tijs van der Storm : Reviewer Science of Computer Programming
- Tijs van der Storm : Reviewer Journal of Systems and Software
- Tijs van der Storm : Co-organizer Lorentz Workshop on Language Interaction Design (LIXD)
- Tijs van der Storm : Co-organizer 1st Dutch Conference on Software Development Automation (SDA'13)
- Tijs van der Storm : Co-organizer CWI Scientific Meetings
- Tijs van der Storm : Organizer Normalized Systems Seminar
- Jurgen Vinju : Member steering committee IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)
- Jurgen Vinju : General Chair IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)
- Jurgen Vinju : Observer IFIP TC2 Working Group 2.3 Programming Methodology

- Jurgen Vinju : Program chair International Workshop on Advanced Software Development Tools and Techniques - (WASDeTT)
- Jurgen Vinju : Program committee International Conference on Model Transformation - (ICMT)
- Jurgen Vinju : Program committee chair WCRE/CSMR Tool Track
- Jurgen Vinju : Organizer International Workshop on Parsing@SLE
- Jurgen Vinju : Program committee WCRE/CSMR ERA track
- Vadim Zaytsev : Steering committee Seminar Series on Advanced Techniques & Tools for Software Evolution (SATToSE)
- Vadim Zaytsev : Program commitee Software Quality Management (SQM)
- Vadim Zaytsev : Program commitee IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)
- Vadim Zaytsev : Program commitee Extreme Modeling Workshop (XM)
- Vadim Zaytsev : Judging commitee ACM Student Research Competition
- Vadim Zaytsev : Program committee co-chair Working Conference on Reverse Engineering (WCRE/CSMR) Tool-track
- Vadim Zaytsev : Reviewer Science of Computer Programming
- Vadim Zaytsev : Workshop co-chair Open and Original Problems in Software Language Engineering (OOPSLE)
- Vadim Zaytsev : Social media co-chair ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS)
- Vadim Zaytsev: Colloquium organiser of Programming Environment Meetings (PEM)

7.2. Teaching - Supervision - Juries

7.2.1. Teaching

Master : Jurgen Vinju, Paul Klint, Software Evolution, 6 EC, Universiteit van Amsterdam, The Netherlands

Master : Tijs van der Storm, Jurgen Vinju, Software Construction, 6 EC, Universiteit van Amsterdam, The Netherlands

Master : Jan van Eijck, Bert Lisser, Vadim Zaytsev, Software Testing, 6 EC, Universiteit van Amsterdam, The Netherlands

Master : Jan van Eijck, Functional Algorithm Specification, 6 EC, Universiteit van Amsterdam, The Netherlands

7.2.2. Supervision

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, supervisor Paul Klint, Tijs van der Storm

PhD in progress : Davy Landman, Recovery and Synthesis of Domain Specific Language Design, started 2011, supervisors Jurgen Vinju, Paul Klint, supervisors Jurgen Vinju .

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

PhD in progress : Ali Afroozeh, Ambiguity and Disambiguation for Context-free Grammars, started 2012, supervisor Jurgen Vinju

PhD in progress : Pablo Inostroza Valdera, Rich UIs for Domain-Specific Languages, started 2013, supervisor Tijs van der Storm

PhD in progress : Ashim Shahi, Principled Quality Assessment of Software, started 2012, supervisor Paul Klint, Jurgen Vinju

PhD in progress : Michael Steindorfer, Scaling meta-programming to data-programming, started 2012, supervisor Paul Klint, Jurgen Vinju

Paul Klint, Jurgen Vinju, Tijs van der Storm, Mark Hills, Jeroen van den Bos and Jan van Eijck together also supervised more than 30 master thesis projects for Universiteit van Amsterdam in 2013.

7.2.3. Juries

- Jan van Eijck, PhD: Arno Bastenhof, Universiteit Utrecht, The Netherlands
- Jan van Eijck, PhD: Frédéric Moisan, Université de Toulouse, France
- Paul Klint, Phd: Romulo Goncalvez, Universiteit van Amsterdam, The Netherlands
- Jurgen Vinju, PhD: C.P.T. de Gouw, Universiteit Leiden, The Netherlands.

7.3. Popularization

- Jan van Eijck : “Why Learn Haskell?”, Talk for UvA Programming Summer School, Amsterdam, July 2, 2013.
- Paul Klint, Davy Landman: “Hoe ontstond de eerste computer?” (“The origin of the first computer”), Public lecture for broad audience, NEMO, Amsterdam.
- Paul Klint, “How to test a meta-program?”, invited talk at the Workshop on Scalable Language Specification (SLS 2013).
- Paul Klint, "BaMa: Key to the Future?”, invited talk at IW1010: 10 Years of UvA Information Sciences, Amsterdam
- Paul Klint, “Understanding the Quality of Open Source Projects”, invited talk at SATToSe, Bern
- Davy Landman : “What Does Control Flow Really Look Like? Eyeballing the Cyclomatic Complexity Metric”, poster and presentation at ICT.OPEN (NWO networking event).
- Michael Steindorfer : “Object Redundancy Profiling in Java”, poster and presentation at ICT.OPEN (NWO networking event).
- Tijs van der Storm : “Domain-specific languages”, Guest lecture Bachelor Computer Science, Universiteit van Amsterdam
- Tijs van der Storm : “Opportunities and Risks of MDD – The case of Derric: a DSL for digital forensics”, Presentation at CodeGeneration 2013.
- Tijs van der Storm : “Implementing Domain-specific languages using Rascal”, Invited Presentation at Sioux: Source of your technology.
- Tijs van der Storm, Kevin van der Vlist, Jimi van der Woning : “Questionnaires in Rascal”, participation Language Workbench Challenge 2013 (LWC’13).
- Tijs van der Storm, Alex Loh, “Questionnaires in Ensō”, participation Language Workbench Challenge 2013 (LWC’13).
- Tijs van der Storm : “QL: a language for questionnaires”, assignment description LWC’13.
- Tijs van der Storm : “Software Development Automation Research: Collaboration with Industry”, presentation at the 1st Dutch conference on Software Development Automation (SDA’13).

- Jurgen Vinju : “Modularity”, Guest lecture Bachelor Computer Science, Universiteit van Amsterdam
- Jurgen Vinju : “Software Analysis and Transformation with Rascal” , Presentation NBIC BioAssist meeting (BioAssist)
- Jurgen Vinju, Tijs van der Storm, Atze van der Ploeg, Anastasia Izmaylova, Ali Afroozeh : CWI Open Day, demonstration of NAO robot programming to children using dedicated DSL “Marvol”.
- Jurgen Vinju, Tijs van der Storm, Atze van der Ploeg : “CWI In Bedrijf” (CWI and Industry) , demonstration of Rascal language workbench using the NAO robot DSL “Marvol”.
- Vadim Zaytsev : “A Snappy Introduction to Metaprogramming in Rascal”, RedDevCon’13.
- Vadim Zaytsev : “Modeling Software Structures with GrammarLab”, Tutorial at MoDELS’13.

8. Bibliography

Major publications by the team in recent years

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

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

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

[4] S. ERDWEG, T. STORM, M. VOELTER, M. BOERSMA, R. BOSMAN, W. R. COOK, A. GERRITSEN, A. HULSHOUT, S. KELLY, A. LOH, G. KONAT, P. J. MOLINA, M. PATATNIK, R. POHJONEN, E. SCHINDLER, K. SCHINDLER, R. SOLMI, V. VERGU, K. B. VLIST, G. WACHSMUTH, J. M. WONING. *The State Of The Art In Language Workbenches. Conclusions From The Language Workbench Challenge*, in "Proceedings of the International Conference on Software Language Engineering (SLE, 2013)", Indianapolis, USA, 2013, http://hal.inria.fr/hal-00923386

[5] M. HILLS, P. KLINT, J. VINJU. *Meta-Language Support For Type-Safe Access To External Resources*, in "Pre-Proceedings of the 5th International Conference on Software Language Engineering", Dresden, Netherlands, K. CZARNECKI, G. HEDIN (editors), Fakultät Informatik, Technische Universität Dresden, 2012, pp. 370 - 389, http://hal.inria.fr/hal-00756878

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

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

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

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

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

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

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

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

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

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

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

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

Publications of the year

Articles in International Peer-Reviewed Journals

[18] F. A. G. SIETSMA, K. R. APT. *Common Knowledge In Email Exchanges*, in "ACM Transactions on Computational Logic", 2013, vol. 14, n^{o} 3, pp. 1 - 23, http://hal.inria.fr/hal-00923396

[19] F. A. G. SIETSMA, D. J. N. EIJCK. *Action Emulation Between Canonical Models*, in "Journal of Philosophical Logic", 2013, vol. 42, n^{o} 6, pp. 905 - 925, http://hal.inria.fr/hal-00923395

[20] A. J. VAN DER PLOEG. *Drawing Non-Layered Tidy Trees In Linear Time*, in "Software – Practice and Experience", 2013, http://hal.inria.fr/hal-00923389

[21] D. J. N. VAN EIJCK, T. VAN DER STORM. *Understanding Information Update In Questionnaires*, in "Science of Computer Programming", 2013, http://hal.inria.fr/hal-00923384

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

International Conferences with Proceedings

[23] A. AFROOZEH, M. G. J. VAN DEN BRAND, A. JOHNSTONE, E. SCOTT, J. J. VINJU. *Safe Specification Of Operator Precedence Rules*, in "Proceedings of the International Conference on Software Language Engineering (SLE, 2013)", K. CZARNECKI, G. HEDIN (editors), 2013, http://hal.inria.fr/hal-00923391

[24] S. ERDWEG, T. VAN DER STORM, M. VOELTER, M. BOERSMA, R. BOSMAN, W. R. COOK, A. GERRITSEN, A. HULSHOUT, S. KELLY, A. LOH, G. KONAT, P. J. MOLINA, M. PALATNIK, R. POHJONEN, E. SCHINDLER, K. SCHINDLER, R. SOLMI, V. VERGU, K. B. VAN DER VLIST, G. WACHSMUTH, J. M. VAN DER WONING. *The State Of The Art In Language Workbenches. Conclusions From The Language Workbench Challenge*, in "Proceedings of the International Conference on Software Language Engineering (SLE, 2013)", 2013, http://hal.inria.fr/hal-00923386

[25] M. A. HILLS, P. KLINT, J. J. VINJU. *An Empirical Study Of PHP Feature Usage*, in "Proceedings of the International Symposium on Software Testing and Analysis (ISSTA, 2013)", M. PEZZE, M. HARMAN (editors), 2013, http://hal.inria.fr/hal-00923390

[26] P. KLINT, D. LANDMAN, J. J. VINJU. *Exploring The Limits Of Domain Model Recovery*, in "29th IEEE International Conference on Software Maintenance (ICSM), 2013", IEEE Computer Society, 2013, pp. 120 - 129, http://hal.inria.fr/hal-00923392

[27] P. KLINT, A. J. VAN DER PLOEG. *A Library For Declarative Resolution-Independent 2D Graphics*, in "Proceedings of the Practical Aspects of Declarative Languages (PADL, 2013)", 2013, http://hal.inria.fr/hal-00923381

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

[29] R. LÄMMEL, V. ZAYTSEV. *Language Support For Megamodel Renarration*, in "Proceedings of the Extreme Modeling Workshop (XM, 2013)", J. DE LARA, D. DI RUSCIO, A. PIERANTONIO (editors), 2013, http://hal.inria.fr/hal-00923398

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

[31] A. J. Van Der Ploeg. *Monadic Functional Reactive Programming*, in "Proceedings of the ACM SIGPLAN Haskell Symposium", C. Shan (editor), 2013, http://hal.inria.fr/hal-00923382

[32] V. Zaytsev. *Micropatterns In Grammars*, in "Proceedings of the International Conference on Software Language Engineering (SLE, 2013)", M. Erwig, R. F. Paige, E. van Wyk (editors), 2013, http://hal.inria.fr/hal-00923399

[33] V. Zaytsev. *Pending Evolution Of Grammars*, in "Proceedings of the Extreme Modeling Workshop (XM, 2013)", J. de Lara, D. Di Ruscio, A. Pierantonio (editors), 2013, http://hal.inria.fr/hal-00923397

[34] J. van den Bos, T. van der Storm. *A Case Study In Evidence-Based DSL Evolution*, in "Proceedings of the 9th European Conference on Modelling Foundations and Applications", P. Van Gorp, T. Ritter, L. M. Rose (editors), Lecture Notes in Computer Science, Springer, 2013, vol. 7949, pp. 207 - 219, http://hal.inria.fr/hal-00923401

[35] J. van den Bos, T. van der Storm. *TRINITY: An IDE For The Matrix*, in "Proceedings of the 28th IEEE International Conference on Software Maintenance", IEEE, 2013, pp. 520 - 523, http://hal.inria.fr/hal-00923400

[36] T. van der Storm. *Semantic Deltas For Live DSL Environments*, in "Proceedings of the International Workshop on Live Programming (LIVE, 2013)", 2013, http://hal.inria.fr/hal-00923388

Research Reports

[37] P. R. Griffioen. , *Type Inference For Linear Algebra With Units Of Measurement*, 2013, n^{O} SwAT-1302, pp. 1 - 32, http://hal.inria.fr/hal-00923380

[38] A. Izmaylova, P. Klint, A. Shahi, J. J. Vinju. , *M3: An Open Model For Measuring Code Artifacts*, 2013, n^{O} arXiv-1312.1188, pp. 1-2, http://hal.inria.fr/hal-00923379