IN PARTNERSHIP WITH:
Université des sciences et technologies de Lille (Lille 1)

Activity Report 2018

Project-Team RMOD

Analyses and Languages Constructs for Object-Oriented Application Evolution

IN COLLABORATION WITH: Centre de Recherche en Informatique, Signal et Automatique de Lille

RESEARCH CENTER
Lille - Nord Europe

THEME
Distributed programming and Software engineering

Table of contents

Project-Team RMOD

Creation of the Project-Team: 2009 July 01

Keywords:

Computer Science and Digital Science:

A2. - Software
A2.1. - Programming Languages
A2.1.3. - Object-oriented programming
A2.1.8. - Aspect-oriented programming
A2.1.10. - Domain-specific languages
A2.1.12. - Dynamic languages
A2.3.1. - Embedded systems
A2.5. - Software engineering
A2.5.1. - Software Architecture & Design
A2.5.3. - Empirical Software Engineering
A2.5.4. - Software Maintenance & Evolution
A2.6. - Infrastructure software
A2.6.3. - Virtual machines

Other Research Topics and Application Domains:

B2. - Health
B2.7. - Medical devices
B5. - Industry of the future
B5.9. - Industrial maintenance
B6.5. - Information systems
B7. - Transport and logistics

1. Team, Visitors, External Collaborators

Research Scientists

Stéphane Ducasse [Team leader, Inria, Senior Researcher, HDR]
Marcus Denker [Inria, Researcher]

Faculty Members

Nicolas Anquetil [Univ des sciences et technologies de Lille, Associate Professor, HDR]
Anne Etien [Univ de Lille, Associate Professor, HDR]
Damien Pollet [Univ de Lille, Associate Professor]
Vincent Aranega [Univ de Lille, Associate Professor, from Sep 2018]

Post-Doctoral Fellow

Henrique Santos Camargos Rocha [Inria]

PhD Students

Lionel Akue [Inria, from Nov 2018]
Julien Delplanque [Univ de Lille]
Thomas Dupriez [Ecole Normale Supérieure Paris, from Oct 2018]
Brice Govin [Thales, 3rd year, granted by CIFRE, until 2018]
Carolina Hernandez Phillips [Inria, from Oct 2018]

Jason Lecerf [CEA]
Matteo Marra [VUB Brussels]
Pablo Tesone [Ecole des Mines de Douai]

Technical staff

Clément Béra [Inria, until Mar 2018]
Steven Costiou [Inria, from Oct 2018]
Cyril Ferlicot-Delbecque [Inria, from Oct 2018]
Pavel Krivanek [Inria]
Denis Kudriashov [Inria, until Apr 2018]
Guillaume Larcheveque [Inria, from May 2018, granted by ANR CARNOT Abondement project]
Allex Oliveira [Inria, from May 2018]
Christophe Demarey [Inria, Engineers, 70%]

Interns

Hamdi Gabsi [ENIS, Tunisia, from Oct 2018 until Nov 2018]
Lionel Akue [Inria, from Jul 2018 until Sep 2018]
Asbathou Biyalou-Sama [Inria, from Apr 2018 until Aug 2018]
Benoît Verhaeghe [Berger Levrault, from March 2018 to Dec 2018]
Quentin Ducasse [Inria, from Jun 2018 until Aug 2018]
Thomas Dupriez [Ecole Normale Supérieure Cachan, until Feb 2018]
Yoan Geran [Ecole Normale Supérieure Paris, from Jun 2018 until Jul 2018]
Myroslava Romaniuk [Inria, from Jul 2018 until Sep 2018]
Pierre Tsapliayeu [Univ de Lille, from Apr 2018 until Aug 2018]
Eleonore Wald [Univ de Lille, from Apr 2018 until Jul 2018]
Oleksandr Zaitsev [Inria, from Oct 2018]

Administrative Assistant

Julie Jonas [Inria]

Visiting Scientists

Abdelghani Alidra [University of Skikda Algeria, from Nov 2018]
Andy Amoordon [Utocat, from Oct 2018]
Jan Bliznicenko [University of Prague , Mar 2018]
Abdelhakim Bouremel [University of Skikda Algeria, May 2018]
Christopher Fuhrman [Ecole de technologie superieure de montreal, from Oct 2018]
Tomohiro Oda [Key Technology Laboratory, Japan, from Aug 2018 until Sep 2018]
Giuseppe Antonio Pierro [University of Cagliari, from Aug 2018]
Ronie Salgado Faila [University of Chile at Santiago, Chile, from Aug 2018 until Sep 2018]
Serge Demeyer [Universiteit Antwerpen, Belgium, from Jun 2018 until Jul 2018]

External Collaborators

Guillermo Polito [CNRS]
Olivier Auverlot [Univ de Lille]
Esteban Lorenzano [Inria Soft]
Santiago Bragagnolo [InriaTech]

2. Overall Objectives

2.1. Introduction

Keywords: Software evolution, Maintenance, Program visualization, Program analyses, Meta modelling, Software metrics, Quality models, Object-oriented programming, Reflective programming, Traits, Dynamically typed languages, Dynamic Software Update, Pharo, Moose.

RMoD's general vision is defined in two objectives: remodularization and modularity constructs. These two views are the two faces of a same coin: maintenance could be eased with better engineering and analysis tools and programming language constructs could let programmers define more modular applications.

2.2. Reengineering and remodularization

While applications must evolve to meet new requirements, few approaches analyze the implications of their original structure (modules, packages, classes) and their transformation to support their evolution. Our research focuses on the *remodularization* of object-oriented applications. Automated approaches including clustering algorithms are not satisfactory because they often ignore user inputs. Our vision is that we need better approaches to support the transformation of existing software. The reengineering challenge tackled by RMoD is formulated as follows:

How to help remodularize existing software applications?

We are developing analyses and algorithms to remodularize object-oriented applications. This is why we started studying and building tools to support the *understanding of applications* at the level of packages and modules. This allows us to understand the results of the *analyses* that we are building.

We seek to create tools to help developers perform large refactoring. How can they keep track of changes in various locations in a system while ensuring *integrity of current and new code* by *uniformally applying new design choices*.

2.3. Constructs for modular and isolating programming languages

Dynamically-typed programming languages such as JavaScript are getting new attention as illustrated by the large investment of Google in the development of the Chrome V8 JavaScript engine and the development of a new dynamic language DART. This new trend is correlated to the increased adoption of dynamic programming languages for web-application development, as illustrated by Ruby on Rails, PHP and JavaScript. With web applications, users expect applications to be always available and getting updated on the fly. This continuous evolution of application is a real challenge [44]. Hot software evolution often requires *reflective* behavior and features. For instance in CLOS and Smalltalk each class modification automatically migrates existing instances on the fly.

At the same time, there is a need for *software isolation i.e.,*, applications should reliably run co-located with other applications in the same virtual machine with neither confidential information leaks nor vulnerabilities. Indeed, often for economical reasons, web servers run multiple applications on the same virtual machine. Users need confined applications. It is important that (1) an application does not access information of other applications running on the same virtual machine and (2) an application authorized to manipulate data cannot pass such authorization or information to other parts of the application that should not get access to it.

Static analysis tools have always been confronted to reflection [41]. Without a full treatment of reflection, static analysis tools are both incomplete and unsound. Incomplete because some parts of the program may not be included in the application call graph, and unsound because the static analysis does not take into account reflective features [50]. In reflective languages such as F-Script, Ruby, Python, Lua, JavaScript, Smalltalk and Java (to a certain extent), it is possible to nearly change any aspect of an application: change objects, change classes dynamically, migrate instances, and even load untrusted code.

Reflection and isolation concerns are a priori antagonistic, pulling language design in two opposite directions. Isolation, on the one hand, pulls towards more static elements and types (*e.g.,*, ownership types). Reflection, on the other hand, pulls towards fully dynamic behavior. This tension is what makes this a real challenge: As experts in reflective programming, dynamic languages and modular systems, we believe that by working on this important tension we can make a breakthrough and propose innovative solutions in resolving or mitigating this tension. With this endeavor, we believe that we are working on a key challenge that can have an impact on future programming languages. The language construct challenge tackled by RMoD is formulated as follows:

What are the language modularity constructs to support isolation?

In parallel we are continuing our research effort on traits [1] by assessing trait scalability and reuse on a large case study and developing a pure trait-based language. In addition, we dedicate efforts to remodularizing a meta-level architecture in the context of the design of an isolating dynamic language. Indeed at the extreme, modules and structural control of reflective features are the first steps towards flexible, dynamic, yet isolating, languages. As a result, we expect to demonstrate that having adequate composable units and scoping units will help the evolution and recomposition of an application.

3. Research Program

3.1. Software Reengineering

Strong coupling among the parts of an application severely hampers its evolution. Therefore, it is crucial to answer the following questions: How to support the substitution of certain parts while limiting the impact on others? How to identify reusable parts? How to modularize an object-oriented application?

Having good classes does not imply a good application layering, absence of cycles between packages and reuse of well-identified parts. Which notion of cohesion makes sense in presence of late-binding and programming frameworks? Indeed, frameworks define a context that can be extended by subclassing or composition: in this case, packages can have a low cohesion without being a problem for evolution. How to obtain algorithms that can be used on real cases? Which criteria should be selected for a given remodularization?

To help us answer these questions, we work on enriching Moose, our reengineering environment, with a new set of analyses [35], [34]. We decompose our approach in three main and potentially overlapping steps:

1. Tools for understanding applications,
2. Remodularization analyses,
3. Software Quality.

3.1.1. Tools for understanding applications

Context and Problems. We are studying the problems raised by the understanding of applications at a larger level of granularity such as packages or modules. We want to develop a set of conceptual tools to support this understanding.

Some approaches based on Formal Concept Analysis (FCA) [63] show that such an analysis can be used to identify modules. However the presented examples are too small and not representative of real code.

Research Agenda.

FCA provides an important approach in software reengineering for software understanding, design anomalies detection and correction, but it suffers from two problems: (i) it produces lattices that must be interpreted by the user according to his/her understanding of the technique and different elements of the graph; and, (ii) the lattice can rapidly become so big that one is overwhelmed by the mass of information and possibilities [24]. We look for solutions to help people putting FCA to real use.

3.1.2. Remodularization analyses

Context and Problems. It is a well-known practice to layer applications with bottom layers being more stable than top layers [51]. Until now, few works have attempted to identify layers in practice: Mudpie [65] is a first cut at identifying cycles between packages as well as package groups potentially representing layers. DSM (dependency structure matrix) [64], [59] seems to be adapted for such a task but there is no serious empirical experience that validates this claim. From the side of remodularization algorithms, many were defined for procedural languages [47]. However, object-oriented programming languages bring some specific problems linked with late-binding and the fact that a package does not have to be systematically cohesive since it can be an extension of another one [66], [38].

[1] Traits are groups of methods that can be composed orthogonally to simple inheritance. Contrary to mixin, the class has the control of the composition and conflict management.

As we are designing and evaluating algorithms and analyses to remodularize applications, we also need a way to understand and assess the results we are obtaining.

Research Agenda. We work on the following items:

Layer identification. We propose an approach to identify layers based on a semi-automatic classification of package and class interrelationships that they contain. However, taking into account the wish or knowledge of the designer or maintainer should be supported.

Cohesion Metric Assessment. We are building a validation framework for cohesion/coupling metrics to determine whether they actually measure what they promise to. We are also compiling a number of traditional metrics for cohesion and coupling quality metrics to evaluate their relevance in a software quality setting.

3.1.3. Software Quality

Research Agenda. Since software quality is fuzzy by definition and a lot of parameters should be taken into account we consider that defining precisely a unique notion of software quality is definitively a Grail in the realm of software engineering. The question is still relevant and important. We work on the two following items:

Quality models. We studied existing quality models and the different options to combine indicators — often, software quality models happily combine metrics, but at the price of losing the explicit relationships between the indicator contributions. There is a need to combine the results of one metric over all the software components of a system, and there is also the need to combine different metric results for any software component. Different combination methods are possible that can give very different results. It is therefore important to understand the characteristics of each method.

Bug prevention. Another aspect of software quality is validating or monitoring the source code to avoid the emergence of well known sources of errors and bugs. We work on how to best identify such common errors, by trying to identify earlier markers of possible errors, or by helping identifying common errors that programmers did in the past.

3.2. Language Constructs for Modular Design

While the previous axis focuses on how to help remodularizing existing software, this second research axis aims at providing new language constructs to build more flexible and recomposable software. We will build on our work on traits [61], [36] and classboxes [25] but also start to work on new areas such as isolation in dynamic languages. We will work on the following points: (1) Traits and (2) Modularization as a support for isolation.

3.2.1. Traits-based program reuse

Context and Problems. Inheritance is well-known and accepted as a mechanism for reuse in object-oriented languages. Unfortunately, due to the coarse granularity of inheritance, it may be difficult to decompose an application into an optimal class hierarchy that maximizes software reuse. Existing schemes based on single inheritance, multiple inheritance, or mixins, all pose numerous problems for reuse.

To overcome these problems, we designed a new composition mechanism called Traits [61], [36]. Traits are pure units of behavior that can be composed to form classes or other traits. The trait composition mechanism is an alternative to multiple or mixin inheritance in which the composer has full control over the trait composition. The result enables more reuse than single inheritance without introducing the drawbacks of multiple or mixin inheritance. Several extensions of the model have been proposed [33], [55], [26], [37] and several type systems were defined [39], [62], [56], [49].

Traits are reusable building blocks that can be explicitly composed to share methods across unrelated class hierarchies. In their original form, traits do not contain state and cannot express visibility control for methods. Two extensions, stateful traits and freezable traits, have been proposed to overcome these limitations. However, these extensions are complex both to use for software developers and to implement for language designers.

Research Agenda: Towards a pure trait language. We plan distinct actions: (1) a large application of traits, (2) assessment of the existing trait models and (3) bootstrapping a pure trait language.

- To evaluate the expressiveness of traits, some hierarchies were refactored, showing code reuse [28]. However, such large refactorings, while valuable, may not exhibit all possible composition problems, since the hierarchies were previously expressed using single inheritance and following certain patterns. We want to redesign from scratch the collection library of Smalltalk (or part of it). Such a redesign should on the one hand demonstrate the added value of traits on a real large and redesigned library and on the other hand foster new ideas for the bootstrapping of a pure trait-based language.

  In particular we want to reconsider the different models proposed (stateless [36], stateful [27], and freezable [37]) and their operators. We will compare these models by (1) implementing a trait-based collection hierarchy, (2) analyzing several existing applications that exhibit the need for traits. Traits may be flattened [54]. This is a fundamental property that confers to traits their simplicity and expressiveness over Eiffel's multiple inheritance. Keeping these aspects is one of our priority in forthcoming enhancements of traits.
- Alternative trait models. This work revisits the problem of adding state and visibility control to traits. Rather than extending the original trait model with additional operations, we use a fundamentally different approach by allowing traits to be lexically nested within other modules. This enables traits to express (shared) state and visibility control by hiding variables or methods in their lexical scope. Although the traits' "flattening property" no longer holds when they can be lexically nested, the combination of traits with lexical nesting results in a simple and more expressive trait model. We formally specify the operational semantics of this combination. Lexically nested traits are fully implemented in AmbientTalk, where they are used among others in the development of a Morphic-like UI framework.
- We want to evaluate how inheritance can be replaced by traits to form a new object model. For this purpose we will design a minimal reflective kernel, inspired first from ObjVlisp [32] then from Smalltalk [42].

3.2.2. Reconciling Dynamic Languages and Isolation

Context and Problems. More and more applications require dynamic behavior such as modification of their own execution (often implemented using reflective features [46]). For example, F-script allows one to script Cocoa Mac-OS X applications and Lua is used in Adobe Photoshop. Now in addition more and more applications are updated on the fly, potentially loading untrusted or broken code, which may be problematic for the system if the application is not properly isolated. Bytecode checking and static code analysis are used to enable isolation, but such approaches do not really work in presence of dynamic languages and reflective features. Therefore there is a tension between the need for flexibility and isolation.

Research Agenda: Isolation in dynamic and reflective languages. To solve this tension, we will work on *Sure*, a language where isolation is provided by construction: as an example, if the language does not offer field access and its reflective facilities are controlled, then the possibility to access and modify private data is controlled. In this context, layering and modularizing the meta-level [29], as well as controlling the access to reflective features [30], [31] are important challenges. We plan to:

- Study the isolation abstractions available in erights (http://www.erights.org) [53], [52], and Java's class loader strategies [48], [43].
- Categorize the different reflective features of languages such as CLOS [45], Python and Smalltalk [57] and identify suitable isolation mechanisms and infrastructure [40].
- Assess different isolation models (access rights, capabilities [58]...) and identify the ones adapted to our context as well as different access and right propagation.
- Define a language based on
  - the decomposition and restructuring of the reflective features [29],

- the use of encapsulation policies as a basis to restrict the interfaces of the controlled objects [60],
- the definition of method modifiers to support controlling encapsulation in the context of dynamic languages.

An open question is whether, instead of providing restricted interfaces, we could use traits to grant additional behavior to specific instances: without trait application, the instances would only exhibit default public behavior, but with additional traits applied, the instances would get extra behavior. We will develop *Sure*, a modular extension of the reflective kernel of Smalltalk (since it is one of the languages offering the largest set of reflective features such as pointer swapping, class changing, class definition...) [57].

4. Application Domains

4.1. Programming Languages and Tools

Many of the results of RMoD are improving programming languages or development tools for such languages. As such the application domain of these results is as varied as the use of programming languages in general. Pharo, the language that RMoD develops, is used for a very broad range of applications. From pure research experiments to real world industrial use (the Pharo Consortium, http://consortium.pharo.org, has more than 25 company members).

Examples are web applications, server backends for mobile applications or even graphical tools and embedded applications

4.2. Software Reengineering

Moose is a language-independent environment for reverse and re-engineering complex software systems. Moose provides a set of services including a common meta-model, metrics evaluation and visualization. As such Moose is used for analyzing software systems to support understanding and continuous development as well as software quality analysis.

5. Highlights of the Year

5.1. Highlights of the Year

5.1.1. Release of Pharo 7

We released a release candidate for Pharo 7, with a release to be expected early 2019. More information at http://pharo.org.

5.1.2. Awards

- Guillermo Polito, Pablo Tesone, Esteban Lorenzano and Nicolás Passerini won the 1st place in the Innovation Technologies Award at ESUG 2018.
- Christian Marlon Souza Couto, Henrique Rocha, and Ricardo Terra. A Quality-oriented Approach to Recommend Move Method Refactorings. 1st place in 17th Brazilian Symposium on Software Quality, SBQS p. 11—20, ACM, New York, NY, USA, 2018.

6. New Software and Platforms

6.1. Moose

Moose: Software and Data Analysis Platform

KEYWORDS: Software engineering - Meta model - Software visualisation
FUNCTIONAL DESCRIPTION: Moose is an extensive platform for software and data analysis. It offers multiple services ranging from importing and parsing data, to modeling, to measuring, querying, mining, and to building interactive and visual analysis tools. The development of Moose has been evaluated to 200 man/year.

Mots-cles : MetaModeling, Program Visualization, Software metrics, Code Duplication, Software analyses, Parsers

- Participants: Anne Etien, Nicolas Anquetil, Olivier Auverlot, Stéphane Ducasse, Julien Delplanque, Guillaume Larcheveque, Cyril Ferlicot-Delbecque and Pavel Krivanek
- Partners: Université de Berne - Sensus - Synectique - Pleiad - USI - Vrije Universiteit Brussel
- Contact: Stéphane Ducasse
- URL: http://www.moosetechnology.org

6.2. Pharo

KEYWORDS: Live programmation objet - Reflective system - Web Application
FUNCTIONAL DESCRIPTION: Pharo is a pure object reflective and dynamic language inspired by Smalltalk. In addition, Pharo comes with a full advanced programming environment developed under the MIT License. It provides a platform for innovative development both in industry and research. By providing a stable and small core system, excellent developer tools, and maintained releases, Pharo's goal is to be a platform to build and deploy mission critical applications, while at the same time continue to evolve. Pharo 60 got 100 contributors world-wide. It is used by around 30 universities, 15 research groups and around 40 companies.

- Participants: Christophe Demarey, Clement Bera, Damien Pollet, Esteban Lorenzano, Marcus Denker, Stéphane Ducasse and Guillermo Polito
- Partners: BetaNine - Reveal - Inceptive - Netstyle - Feenk - ObjectProfile - GemTalk Systems - Greyc Université de Caen - Basse-Normandie - Université de Berne - Yesplan - RMod - Pleiad - Sensus - Université de Bretagne Occidentale - École des Mines de Douai - ENSTA - Uqbar foundation Argentina - LAM Research - ZWEIDENKER - LifeWare - JPMorgan Chase - KnowRoaming - ENIT - Spesenfuchs - FINWorks - Esug - FAST - Ingenieubüro Schmidt - Projector Software - HRWorks - Inspired.org - Palantir Solutions - High Octane - Soops - Osoco - Ta Mère SCRL - University of Yaounde 1 - Software Quality Laboratory, University of Novi Sad - Software Institute Università della Svizzera italiana - Universdad Nacional de Quilmes - UMMISCO IRD - Université technique de Prague
- Contact: Marcus Denker
- URL: http://www.pharo.org

6.3. Pillar

KEYWORDS: HTML - LaTeX - HTML5
FUNCTIONAL DESCRIPTION: Pillar is a markup syntax and associated tools to write and generate documentation and books. Pillar is currently used to write several books and other documentation. It is used in the tools developed by Feenk.com.

- Partner: Feenk
- Contact: Stéphane Ducasse
- URL: https://github.com/Pillar-markup/pillar

7. New Results

7.1. Dynamic Languages: Virtual Machines

Assessing primitives performance on multi-stage execution. Virtual machines, besides the interpreter and just-in-time compiler optimization facilities, also include a set of primitive operations that the client language can use. Some of these are essential and cannot be performed in any other way. Others are optional: they can be expressed in the client language but are often implemented in the virtual machine to improve performance when the just-in-time compiler is unable to do so (start-up performance, speculative optimizations not implemented or not mature enough, etc.). In a hybrid runtime, where code is executed by an interpreter and a just-in-time compiler, the implementor can choose to implement optional primitives in the client language , in the virtual machine implementation language (typically C or C++), or on top of the just-in-time compiler back-end. This raises the question of the maintenance and performance trade-offs of the different alternatives. As a case study, we implemented the String comparison optional primitive in each case. The paper describes the different implementations, discusses the maintenance cost of each of them and evaluates for different string sizes the execution time in Cog, a Smalltalk virtual machine. [18]

Fully Reflective Execution Environments: Virtual Machines for More Flexible Software. VMs are complex pieces of software that implement programming language semantics in an efficient, portable, and secure way. Unfortunately, mainstream VMs provide applications with few mechanisms to alter execution semantics or memory management at run time. We argue that this limits the evolvability and maintainability of running systems for both, the application domain, e.g., to support unforeseen requirements, and the VM domain, e.g., to modify the organization of objects in memory. This work explores the idea of incorporating reflective capabilities into the VM domain and analyzes its impact in the context of software adaptation tasks. We characterize the notion of a fully reflective VM, a kind of VM that provides means for its own observability and modifiability at run time. This enables programming languages to adapt the underlying VM to changing requirements. We propose a reference architecture for such VMs and present TruffleMATE as a prototype for this architecture. We evaluate the mechanisms TruffleMATE provides to deal with unanticipated dynamic adaptation scenarios for security, optimization, and profiling aspects. In contrast to existing alternatives, we observe that TruffleMATE is able to handle all scenarios, using less than 50 lines of code for each, and without interfering with the application's logic. [2]

7.2. Dynamic Languages: Language Constructs for Modular Design

Dynamic Software Update from Development to Production. Dynamic Software Update (DSU) solutions update applications while they are executing. These solutions are typically used in production to minimize application downtime, or in integrated development environments to provide live programming support. Each of these scenarios presents different challenges, forcing existing solutions to be designed with only one of these use cases in mind. For example, DSUs for live programming typically do not implement safe point detection or instance migration, while production DSUs require manual generation of patches and lack IDE integration. Also, these solutions have limited ability to update themselves or the language core libraries, and some of them present execution penalties outside the update window. We propose a DSU (gDSU) that works for both live programming and production environments. Our solution implements safe update point detection using call stack manipulation and a reusable instance migration mechanism to minimize manual intervention in patch generation. Moreover, it also offers updates of core language libraries and the update mechanism itself. This is achieved by the incremental copy of the modified objects and an atomic commit operation. We show that our solution does not affect the global performance of the application and it presents only a run-time penalty during the update window. Our solution is able to apply an update impacting 100,000 instances in 1 second. In this 1 second, only during 250 milliseconds the application is not responsive. The rest of the time the application runs normally while gDSU is looking for the safe update point. The update only requires to copy the elements that are modified. [6]

Implementing Modular Class-based Reuse Mechanisms on Top of a Single Inheritance VM. Code reuse is a good strategy to avoid code duplication and speed up software development. Existing object-oriented programming languages propose different ways of combining existing and new code such as e.g., single inheritance, multiple inheritance, Traits or Mixins. All these mechanisms present advantages and disadvantages and there are situations that require the use of one over the other. To avoid the complexity of implementing a virtual machine (VM), many of these mechanisms are often implemented on top of an existing high-performance VM, originally meant to run a single inheritance object-oriented language. These implementations require thus a mapping between the programming model they propose and the execution model provided by the VM. Moreover, reuse mechanisms are not usually composable, nor it is easy to implement new ones for a given language. We propose a modular meta-level runtime architecture to implement and combine different code reuse mechanisms. This architecture supports dynamic combination of several mechanisms without affecting runtime performance in a single inheritance object-oriented VM. It includes moreover a reflective Meta-Object Protocol to query and modify classes using the programming logical model instead of the underlying low-level runtime model. Thanks to this architecture, we implemented Stateful Traits, Mixins, CLOS multiple inheritance, CLOS Standard Method Combinations and Beta prefixing in a modular and composable way. [15]

7.3. Software Reengineering

A Reflexive and Automated Approach to Syntactic Pattern Matching in Code Transformations. Empowering software engineers often requires to let them write code transformations. However existing automated or tool-supported approaches force developers to have a detailed knowledge of the internal representation of the underlying tool. While this knowledge is time consuming to master, the syntax of the language, on the other hand, is already well known to developers and can serve as a strong foundation for pattern matching. Pattern languages with metavariables (that is variables holding abstract syntax subtrees once the pattern has been matched) have been used to help programmers define program transformations at the language syntax level. The question raised is then the engineering cost of metavariable support. Our contribution is to show that, with a GLR parser, such patterns with metavariables can be supported by using a form of runtime reflexivity on the parser internal structures. This approach allows one to directly implement such patterns on any parser generated by a parser generation framework, without asking the pattern writer to learn the AST structure and node types. As a use case for that approach we describe the implementation built on top of the SmaCC (Smalltalk Compiler Compiler) GLR parser generator framework. This approach has been used in production for source code transformations on a large scale. We will express perspectives to adapt this approach to other types of parsing technologies. [12]

Relational Database Schema Evolution: An Industrial Case Study. Modern relational database management systems provide advanced features allowing, for example, to include behavior directly inside the database (stored procedures). These features raise new difficulties when a database needs to evolve (e.g. adding a new table). To get a better understanding of these difficulties, we recorded and studied the actions of a database architect during a complex evolution of the database at the core of a software system. From our analysis, problems faced by the database architect are extracted, generalized and explored through the prism of software engineering. Six problems are identified: (1) difficulty in analyzing and visualizing dependencies between database's entities, (2) difficulty in evaluating the impact of a modification on the database, (3) replicating the evolution of the database schema on other instances of the database, (4) difficulty in testing database's functionalities, (5) lack of synchronization between the IDE's internal model of the database and the database actual state and (6) absence of an integrated tool enabling the architect to search for dependencies between entities, generate a patch or access up to date PostgreSQL documentation. We suggest that techniques developed by the software engineering community could be adapted to help in the development and evolution of relational databases. [10]

A Quality-oriented Approach to Recommend Move Method Refactorings. Refactoring is an important activity to improve software internal structure. Even though there are many refactoring approaches, very few consider their impact on the software quality. We propose a software refactoring approach based on quality

attributes. We rely on the measurements of the Quality Model for Object Oriented Design (QMOOD) to recommend Move Method refactorings that improve software quality. In a nutshell, given a software system S, our approach recommends a sequence of refactorings R1,R2,...,Rn that result in system versions S1, S2,..., Sn, where quality (Si+1) > quality (Si). We empirically calibrated our approach, using four systems, to find the best criteria to measure the quality improvement. We performed three types of evaluation to verify the usefulness of our implemented tool, named QMove. First, we applied our approach on 13 open-source systems achieving an average recall of 84.2%. Second, we compared QMove with two state-of-art refactoring tools (JMove and JDeodorant) on the 13 previously evaluated systems, and QMove showed better recall, precision, and f-score values than the others. Third, we evaluated QMove, JMove, and JDeodorant in a real scenario with two proprietary systems on the eyes of their software architects. As result, the experts positively evaluated a greater number of QMove recommendations. [14]

7.4. Dynamic Languages: Debugging

Collectors. Observing and modifying object-oriented programs often means interacting with objects. At runtime, it can be a complex task to identify those objects due to the live state of the program. Some objects may exist for only a very limited period of time, others can be hardly reachable because they are never stored in variables. To address this problem we present Collectors. They are dedicated objects which can collect objects of interest at runtime and present them to the developer. Collectors are non-intrusive, removable code instrumentations. They can be dynamically specified and injected at runtime. They expose an API to allow their specification and the access to the collected objects. We present an implementation of Collectors in Pharo, a Smalltalk dialect. We enrich the Pharo programming and debugging environment with tools that support the Collectors API. We illustrate the use of these API and tools through the collection and the logging of specific objects in a running IOT application. [9]

Rotten Green Tests: a First Analysis. Unit tests are a tenant of agile programming methodologies, and are widely used to improve code quality and prevent code regression. A passing (green) test is usually taken as a robust sign that the code under test is valid. However, we have noticed that some green tests contain assertions that are never executed; these tests pass not because they assert properties that are true, but because they assert nothing at all. We call such tests Rotten Green Tests. Rotten Green Tests represent a worst case: they report that the code under test is valid, but in fact do nothing to test that validity, beyond checking that the code does not crash. We describe an approach to identify rotten green tests by combining simple static and dynamic analyses. Our approach takes into account test helper methods, inherited helpers, and trait compositions, and has been implemented in a tool called DrTest. We have applied DrTest to several test suites in Pharo 7.0, and identified many rotten tests, including some that have been sleeping in Pharo for at least 5 years. [22]

Mining inline cache data to order inferred types in dynamic languages. The lack of static type information in dynamically-typed languages often poses obstacles for developers. Type inference algorithms can help, but inferring precise type information requires complex algorithms that are often slow. A simple approach that considers only the locally used interface of variables can identify potential classes for variables, but popular interfaces can generate a large number of false positives. We propose an approach called inline-cache type inference (ICTI) to augment the precision of fast and simple type inference algorithms. ICTI uses type information available in the inline caches during multiple software runs, to provide a ranked list of possible classes that most likely represent a variable's type. We evaluate ICTI through a proof-of-concept that we implement in Pharo Smalltalk. The analysis of the top-n+2 inferred types (where n is the number of recorded run-time types for a variable) for 5486 variables from four different software systems shows that ICTI produces promising results for about 75% of the variables. For more than 90% of variables, the correct run-time type is present among first six inferred types. Our ordering shows a twofold improvement when compared with the unordered basic approach, i.e., for a significant number of variables for which the basic approach offered ambiguous results, ICTI was able to promote the correct type to the top of the list. [22]

7.5. Blockchain

Ethereum Query Language Blockchains store a massive amount of heterogeneous data which will only grow in time. When searching for data on the Ethereum platform, one is required to either access the records (blocks) directly by using a unique identifier, or sequentially search several records to find the desired information. Therefore, we propose the Ethereum Query Language (EQL), a query language that allows users to retrieve information from the blockchain by writing SQL-like queries. The queries provide a rich syntax to specify data elements to search information scattered through several records. We claim that EQL makes it easier to search, acquire, format, and present information from the blockchain. [7]

SmartInspect: solidity smart contract inspector. Solidity is a language used for smart contracts on the Ethereum blockchain. Smart contracts are embedded procedures stored with the data they act upon. Debugging smart contracts is a really difficult task since once deployed, the code cannot be reexecuted and inspecting a simple attribute is not easily possible because data is encoded. We address the lack of inspectability of a deployed contract by analyzing contract state using decompilation techniques driven by the contract structure definition. Our solution, SmartInspect, also uses a mirror-based architecture to represent locally object responsible for the interpretation of the contract state. SmartInspect allows contract developers to better visualize and understand the contract stored state without needing to redeploy, nor develop any ad-hoc code. [8]

Preliminary Steps Towards Modeling Blockchain Oriented Software Even though blockchain is mostly popular for its cryptocurrency, smart contracts have become a very prominent blockchain application. Smart contracts are like classes that can be called by client applications outside the blockchain. Therefore it is possible to develop blockchain-oriented software (BOS) that implements part of the business logic in the blockchain by using smart contracts. Currently, there is no design standard to model BOS. Since modeling is an important part of designing a software, developers may struggle to plan their BOS. We show three complementary modeling approaches based on well-known software engineering models and apply them to a BOS example. Our goal is to start the discussion on specialized blockchain modeling notations. [13]

SmartAnvil: Open-Source Tool Suite for Smart Contract Analysis. Smart contracts are new computational units with special properties: they act as classes with aspectual concerns; their memory structure is more complex than mere objects; they are obscure in the sense that once deployed it is difficult to access their internal state; they reside in an append-only chain. There is a need to support the building of new generation tools to help developers. Such support should tackle several important aspects: (1) the static structure of the contract, (2) the object nature of published contracts, and (3) the overall data chain composed of blocks and transactions. In this chapter, we present SmartAnvil an open platform to build software analysis tools around smart contracts. We illustrate the general components and we focus on three important aspects: support for static analysis of Solidity smart contracts, deployed smart contract binary analysis through inspection, and blockchain navigation and querying. SmartAnvil is open-source and supports a bridge to the Moose data and software analysis platform. [21]

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Contracts with Industry

8.1.1. BlockChain

Participants: Henrique Rocha, Marcus Denker, Stéphane Ducasse
From 2016, ongoing.

We started a new collaboration with a local startup (UTOCAT) about tools and languages in the context of Blockchain systems. The collaboration started with a 2 month exploration phase involving an engineer at Inria Tech. A postdoc started in 2017.

8.1.2. Pharo Consortium

Participants: Esteban Lorenzano, Clément Béra, Marcus Denker, Stéphane Ducasse

From 2012, ongoing.

The Pharo Consortium was founded in 2012 and is growing constantly. By the end 2018, it has 32 company members, 17 academic partners. Inria supports the consortium with one full time engineer starting in 2011. In 2018, the Pharo Consortium joined InriaSoft.

More at http://consortium.pharo.org.

8.2. Bilateral Grants with Industry

8.2.1. Thales CIFRE

Participants: Brice Govin, Anne Etien, Nicolas Anquetil, Stéphane Ducasse
From 2015, ongoing.

We are working on large industrial project rearchitecturization. PhD in progress: Brice Govin, *Support to implement a rejuvenated software architecture in legacy software*. CIFRE Thales started Jan 2015.

8.2.2. Remodularization of Architecture

Participants: Nicolas Anquetil, Santiago Bragagnolo Stéphane Ducasse, Anne Etien, Benoît Verhaeghe
From 2017, ongoing.

We started a new collaboration with the software editor Berger Levrault about software architecture remodularization. The collaboration started with an end study project exploring the architecture used in the company in order to later migrate from GWT to Angular JS since GWT will not be backward supported anymore in the next versions. An internship and a PhD CIFRE thesis will start in 2018.

9. Partnerships and Cooperations

9.1. Regional Initiatives

9.1.1. CAR IMT Douai

Participants: Pablo Tesone, Guillermo Polito, Marcus Denker, Stéphane Ducasse with: L. Fabresse and N. Bouraqadi (IMT Douai)
From 2009, ongoing.

We have signed a convention with the CAR team led by Noury Bouraqadi of IMT Douai. In this context we co-supervised three PhD students (Mariano Martinez-Peck, Nick Papoylias and Guillermo Polito). The team is also an important contributor and supporting organization of the Pharo project.

Currently, Pablo Tesone is doing a PhD co-supervided by RMOD and Pr. L. Fabresse and N. Bouraqadi. We are collaborating in the Context of CPER Data since 2018.

9.1.2. CPER DATA

Participants: Marcus Denker, Stéphane Ducasse, Allex Oliveira with: L. Fabresse and N. Bouraqadi (IMT Douai)
From 2018, ongoing.

Funding to work one year on the PharoThings Platform. We are creating content for a website and a Demo in collaboration with IMT Douai.

9.2. National Initiatives

9.2.1. CEA List

Participants: Jason Lecerf, Stéphane Ducasse with T. Goubier (CEA List)

From 2016, ongoing.

Jason Lecerf started a shared PhD Oct 2016: *Reuse of code artifacts for embedded systems through refactoring*.

9.3. European Initiatives

9.3.1. Collaborations in European Programs, Except FP7 & H2020

Namur University, Belgium
Participants: Anne Etien, Nicolas Anquetil, Olivier Auverlot, Stéphane Ducasse.

From Sept 2016 to Dec. 2018.

Lille Nord Europe European Associated Team with the PreCISE research center of Pr. A. Cleve from Namur University (Belgium).

This project aims to study the co-evolution between database structure and programs and to propose recommendations to perform required changes on cascade. These programs are either internal to the schema as functions or triggers or external as applications written in Java or Php built on top of the DB. Our intuition is that software engineering techniques can be efficient for such issues. This project also aims to unify the abstract representation of the DB and its relationships with the internal or external program.

VUB Brussels, Belgium
Participants: Guillermo Polito, Stéphane Ducasse.

From 2016, ongoing.

Student: Matteo Marra, collaboration with Eliza Gonzalez Boix. Guillermo Polito co-supervised Matteo Marra's master thesis.

University of Prague
Participants: Stéphane Ducasse.

From 2015, ongoing.

We are working with Dr. Robert Pergl from the University of Prague. Stéphane Ducasse gave a lecture at the University of Prague in 2018.

9.4. International Initiatives

9.4.1. Inria International Partners

9.4.1.1. Informal International Partners

Uqbar Argentina

Participants: Pablo Tesone, Esteban Lorenzano, Guillermo Polito, Stéphane Ducasse.

From 2015, ongoing.

We are working with the Uqbar team from different Argentinian universities. We hired three of the people: Nicolas Passerini(engineer), Esteban Lorenzano (engineer) and Pablo Tesone (PhD).

Pharo in Research:

Participants: Pablo Tesone, Esteban Lorenzano, Guillermo Polito, Marcus Denker, Stéphane Ducasse.
From 2009, ongoing.

We are building an ecosystem around Pharo with international research groups, universities and companies. Several research groups (such as Software Composition Group – Bern, and Pleaid – Santiago) are using Pharo. Many universities are teaching OOP using Pharo and its books. Several companies worldwide are deploying business solutions using Pharo.

9.5. International Research Visitors

9.5.1. Visits of International Scientists

- Abdelghani Alidra [University of Skikda Algeria, from Nov 2018]
- Andy Amoordon [Utocat, from Oct 2018]
- Jan Bliznicenko [University of Prague , Mar 2018]
- Abdelhakim Bouremel [University of Skikda Algeria, May 2018]
- Thomas Dupriez [Ecole Normale Supŕieure Paris, Sep 2018]
- Christopher Fuhrman [Ecole de technologie superieure de montreal, from Oct 2018]
- Tomohiro Oda [Key Technology Laboratory, Japan, from Aug 2018 until Sep 2018]
- Giuseppe Antonio Pierro [University of Cagliari, from Aug 2018]
- Ronie Salgado Faila [niversity of Chile at Santiago, Chile, from Aug 2018 until Sep 2018]
- Serge Demeyer [Universiteit Antwerpen, Belgium, from Jun 2018 until Jul 2018

9.5.1.1. Internships

- Lionel Akue [Inria, from Jul 2018 until Sep 2018]
- Asbathou Biyalou Sama [Inria, from Apr 2018 until Aug 2018]
- Quentin Ducasse [Inria, from Jun 2018 until Aug 2018]
- Thomas Dupriez [Ecole Normale Supŕieure Cachan, until Feb 2018]
- Yoan Geran [Ecole Normale Supŕieure Paris, from Jun 2018 until Jul 2018]
- Pierre Tsapliayeu [Univ de Lille, from Apr 2018 until Aug 2018]
- Eleonore Wald [Univ de Lille, from Apr 2018 until Jul 2018]
- Oleksandr Zaitsev [Inria, from Oct 2018]
- Myroslava Romaniuk [Inria, from Jul 2018 until Sep 2018]

9.5.2. Visits to International Teams

- Marcus Denker: VUB Brussels in spring and fall 2018 (Lecture).

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Organisation

10.1.1.1. Member of the Organizing Committees

- Marcus Denker and Stéphane Ducasse are in the board of ESUG and organized ESUG 2018, the yearly Smalltalk conference that brings together research and industry (http://www.esug.org/).

10.1.2. Scientific Events Selection

10.1.2.1. Chair of Conference Program Committees

Anne Etien has been PC chair of IWST since 2015.

10.1.2.2. Reviewer

Nicolas Anquetil: SCAM'18, the 18th IEEE International Working Conference on Source Code Analysis and Manipulation.

10.1.3. Journal

10.1.3.1. Reviewer - Reviewing Activities

Nicolas Anquetil reviewed articles for the following international journals:

- IEEE Transactions on Software Engineering;
- Information and Software Technology;
- Journal of Systems and Software.

10.1.4. Scientific Expertise

- Anne Etien: expert for the French government on Research tax credit.
- Marcus Denker: reviewer for EUREKA.

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Master : Christophe Demarey, Intégration continue, 16 EdTD, M2, Université de Lille, France

Licence: Anne Etien, Bases de données, 30h, L3, Polytech-Lille, France

Licence: Anne Etien, Programmation par objet, 40h, L3, Polytech-Lille, France

Master: Anne Etien, Metamodelisation for Reverse Engineering, 5h, M2, Université de Montpellier, France

Master: Anne Etien, Metamodelisation for Reverse Engineering, 12h, M2, Université Paris 1, France

Master: Anne Etien, Metamodelisation for Reverse Engineering, 5h, M2, Tunis, Tunisie

Master: Anne Etien, Test et Maintenance, 10h, M2, Polytech-Lille, France

Master: Anne Etien, Test et Maintenance, 14h, M2, Polytech-Lille, France

Master: Anne Etien, Syste`me d'information objet, 10h, M1, Polytech Lille, France

Master: Anne Etien, Bases de donnés Avancés, 20h, M1, Polytech-Lille, France

Master: Anne Etien, Qualité logicielle, 8h, M2, Université Lille 1, France

Licence : Vincent Aranega, Initiation à l'informatique, 51 EdTD, L1, Université de Lille, France

Licence : Vincent Aranega, Architecture des ordinateurs, 42 EdTD, L2, Université de Lille, France

Licence : Vincent Aranega, Génie Logiciel, 42 EdTD, M1, Université de Lille, France

License : Damien Pollet, OpenDevs, 35h, L3, Université de Lille, France

License : Damien Pollet, Programmation objet en Java, 155h, L3, IMT Lille-Douai, France

License : Damien Pollet, Systèmes numériques, 14h, L3, IMT Lille-Douai, France

Master : Damien Pollet, Technologies des systèmes d'informations, 21h, M1, IMT Lille-Douai, France

Master : Damien Pollet, Ingénierie du logiciel, 7h, M2, IMT Lille-Douai, France

Master : Damien Pollet, Algorithmes pour les réseaux, 27h, M2, IMT Lille-Douai, France

Master: Meta modeling, Ecole des Mines de Douai, 22 EdTD, M1, Nov 2018, Douai, France

Master: Advanced Design, ENIS M1, Dec 2017/2018, Tunis, 22 EdTD, Tunisia

License: Object-oriented programming, Prague L3, Dec 2017/2018, 16 EdTD, Prague, Tcheque Republic

Master: Meta modeling, Pantheon Sorbonne, 27 EdTD, M2, Dec 2018, Paris, France

Master: Meta programming, Universite de Bretagne Occidental, 9 EdTD, brest, France.

Licence : Thomas Dupriez, Mathématiques Discrètes, 39 heures en équivalent TD, L2, Université Lille, France

Master : Marcus Denker, 2 hours (spring and fall each), Advanced Reflection. MetaLinks, VUB Brussels, Belgium.

License: open-dev L3 option 17 hours Lille, France

License: Introduction à la Programmation L3 option 31.5 hours Paul Sabatier Toulouse, France

E-learning

Pharo Mooc, 7 weeks, Licences and Master students

Pedagogical resources: Books, Learning Object-Oriented Programming, Design and TDD with Pharo, A simple reflective object kernel.

10.2.2. Supervision

PhD: Brice Govin, *Support to Implement a Rejuvenated Software Architecture in Legacy Software* CIFRE Thales, 26 June 2018, Anne Etien, Nicolas Anquetil. [1]

PhD: Pablo Tesone, *Hot Software Update In Robotics Applications*, IMT Lille-Douai, 16 December, Luc Fabresse, Stéphane Ducasse

PhD in progress: Jason Lecerf, *Reuse of Code Artifacts for Embedded Systems Through Refactoring*, started Oct 2016, CEA Thierry Goubier, Stéphane Ducasse

PhD in progress: Julien Delplanque, *Software Engineering Techniques Applied to Databases*, started Oct 2017, Anne Etien, Nicolas Anquetil

PhD in progress: Lionel Akue, *Fortran Analysis*, started Oct 2018, Anne Etien, Nicolas Anquetil

PhD in progress: Thomas Dupriez, *New Generation Debugger and Application Monitoring*, started Oct 2018, Stephane Ducasse, Guillermo Polito

PhD in progress: Houekpetodji Mahugnon Honore, *Multi-Facet Actionable for Information System Rejuvenation*, SPI Lille, France, Stephane Ducasse, Nicolas Anquetil, Nicolas Dias, Jerome Sudich

PhD in progress: Carolina Hernandez, *Tools for MicroKernels* Guillermo Polito and Luc Fabresse

10.3. Popularization

10.3.1. Articles and contents

- Stéphane Ducasse and Guillermo Polito. Stéphane Ducasse (Ed.). Physche: A Little Scheme in Pharo, p. 50, Square Bracket Associates, 2018. [20]
- Stéphane Ducasse. Stéphane Ducasse (Ed.). A Simple Reflective Object Kernel, p. 40, Square Bracket Associates, 2018. [19]

10.3.2. Education

- A MOOC for Pharo is online (Stéphane Ducasse).
  http://mooc.pharo.org

10.3.3. Interventions

- Multiple public Pharo Sprints in Lille.
  https://association.pharo.org/events
- IoT Hackathon. One Day Event in Cologne, Germany. Using PharoIoT
  http://zweidenker.de/de/iot-hackathon-2018
  Video: https://www.youtube.com/watch?v=dIl9FAatKyw
- RMOD co-organized and participated at ESUG 2018.
  http://www.esug.org/wiki/pier/Conferences/2018
  - Presentation about PharoIoT.
    https://www.slideshare.net/MarcusDenker/pharo-iot

11. Bibliography

Publications of the year

Doctoral Dissertations and Habilitation Theses

[1] B. GOVIN. *Supporting Legacy Software Architecture Renovation : A real Case by Thales Air Systems*, Lille, June 2018, https://hal.inria.fr/tel-01881319

Articles in International Peer-Reviewed Journals

[2] G. CHARI, D. GARBERVETSKY, S. MARR, S. DUCASSE. *Fully Reflective Execution Environments: Virtual Machines for More Flexible Software*, in "IEEE Transactions on Software Engineering", May 2018, pp. 1 - 20 [*DOI :* 10.1109/TSE.2018.2812715], https://hal.inria.fr/hal-01728111

[3] A. ETIEN, J. LAVAL. *Advances in Dynamic Languages*, in "Science of Computer Programming", September 2018, vol. 161, 1 p. , cited By 0 [*DOI :* 10.1016/J.SCICO.2018.03.004], https://hal.archives-ouvertes.fr/hal-01858483

[4] M. MARRA, G. POLITO, E. GONZALEZ BOIX. *Out-Of-Place debugging: a debugging architecture to reduce debugging interference*, in "The Art, Science, and Engineering of Programming", November 2018, vol. 3, n^{o} 2, pp. 1-29 [*DOI :* 2019/3/3], https://hal.inria.fr/hal-01952790

[5] N. MILOJKOVIĆ, C. BÉRA, M. GHAFARI, O. NIERSTRASZ. *Mining inline cache data to order inferred types in dynamic languages*, in "Science of Computer Programming", September 2018, vol. 161, pp. 105-121 [*DOI :* 10.1016/J.SCICO.2017.11.003], https://hal.inria.fr/hal-01666541

[6] P. TESONE, G. POLITO, N. BOURAQADI, S. DUCASSE, L. FABRESSE. *Dynamic Software Update from Development to Production*, in "The Journal of Object Technology", November 2018, vol. 17, n^{o} 1, pp. 1-36 [*DOI :* 10.5381/JOT.2018.17.1.A2], https://hal.inria.fr/hal-01920362

International Conferences with Proceedings

[7] S. BRAGAGNOLO, H. S. C. ROCHA, M. DENKER, S. DUCASSE. *Ethereum Query Language*, in "WETSEB 2018 - 1st International Workshop on Emerging Trends in Software Engineering for Blockchain", Gothenburg, Sweden, May 2018 [*DOI :* 10.1145/3194113.3194114], https://hal.inria.fr/hal-01831084

[8] S. BRAGAGNOLO, H. S. C. ROCHA, M. DENKER, S. DUCASSE. *SmartInspect: Solidity Smart Contract Inspector*, in "IWBOSE 2018 - 1st International Workshop on Blockchain Oriented Software Engineering", Campobasso, Italy, IEEE, March 2018 [*DOI :* 10.1109/IWBOSE.2018.8327566], https://hal.inria.fr/hal-01831075

[9] S. COSTIOU, M. KERBOEUF, A. PLANTEC, M. DENKER. *Collectors*, in "Programming Experience 2018 (PX'18)", Nice, France, Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, ACM Press, April 2018, 9 p. [*DOI :* 10.1145/3191697.3214335], https://hal.univ-brest.fr/hal-01829183

[10] J. DELPLANQUE, A. ETIEN, N. ANQUETIL, O. AUVERLOT. *Relational Database Schema Evolution: An Industrial Case Study*, in "ICSME 2018 - 34th IEEE International Conference on Software Maintenance

and Evolution", Madrid, Spain, September 2018 [*DOI :* 10.1109/ICSME.2018.00073], https://hal.archives-ouvertes.fr/hal-01945042

[11] S. DEMEYER, B. VERHAEGHE, A. ETIEN, N. ANQUETIL, S. DUCASSE. *Evaluating the Efficiency of Continuous Testing during Test-Driven Development*, in "VST 2018 - 2nd IEEE International Workshop on Validation, Analysis and Evolution of Software Tests", Campobasso, Italy, March 2018, pp. 1-5 [*DOI :* 10.1109/VST.2018.8327152], https://hal.inria.fr/hal-01717343

[12] J. LECERF, J. BRANT, T. GOUBIER, S. DUCASSE. *A Reflexive and Automated Approach to Syntactic Pattern Matching in Code Transformations*, in "ICSME 2018 - 34th IEEE International Conference on Software Maintenance and Evolution", Madrid, Spain, ICSME 2018 - 34th IEEE International Conference on Software Maintenance and Evolution, September 2018 [*DOI :* 10.1109/ICSME.2018.00052], https://hal.archives-ouvertes.fr/hal-01851857

[13] H. ROCHA, S. DUCASSE. *Preliminary Steps Towards Modeling Blockchain Oriented Software*, in "WETSEB 2018 - 1st International Workshop on Emerging Trends in Software Engineering for Blockchain", Gothenburg, Sweden, May 2018 [*DOI :* 10.1145/3194113.3194123], https://hal.inria.fr/hal-01831046

[14] C. M. SOUZA COUTO, H. ROCHA, R. TERRA. *A Quality-oriented Approach to Recommend Move Method Refactorings*, in "Proceedings of the 17th Brazilian Symposium on Software Quality", Curitiba, Brazil, October 2018 [*DOI :* 10.1145/3275245.3275247], https://hal.inria.fr/hal-01944493

[15] P. TESONE, G. POLITO, L. FABRESSE, N. BOURAQADI, S. DUCASSE. *Implementing Modular Class-based Reuse Mechanisms on Top of a Single Inheritance VM*, in "SAC 2018:- The 33rd ACM/SIGAPP Symposium On Applied Computing", Pau, France, SAC 2018 Symposium on Applied Computing, ACM New York, NY, USA, April 2018 [*DOI :* 10.1145/3167132.3167244], https://hal.archives-ouvertes.fr/hal-01812612

[16] M. VIGGIATO, R. TERRA, H. S. C. ROCHA, M. TULIO VALENTE, E. FIGUEIREDO. *Microservices in Practice: A Survey Study*, in "VEM 2018 - 6th Workshop on Software Visualization, Evolution and Maintenance", Sao Carlos, Brazil, September 2018, https://hal.inria.fr/hal-01944464

Conferences without Proceedings

[17] J. DELPLANQUE, O. AUVERLOT, A. ETIEN, N. ANQUETIL. *Définition et identification des tables de nomenclatures*, in "INFORSID 2018 - 36ème édition d'INFormatique des ORganisations et Systèmes d'Information et de Décision", Nantes, France, May 2018, https://hal.archives-ouvertes.fr/hal-01944135

[18] S. KALEBA, C. BÉRA, S. DUCASSE. *Assessing primitives performance on multi-stage execution*, in "ICOOOLPS 2017 - 12th Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems", Amsterdam, Netherlands, July 2018, https://hal.archives-ouvertes.fr/hal-01874946

Scientific Books (or Scientific Book chapters)

[19] S. DUCASSE. *A simple reflective object kernel*, Published by the author, September 2018, https://hal.inria.fr/hal-01900323

[20] S. DUCASSE, G. POLITO. *Physche: A Little Scheme in Pharo*, Published by the authors, August 2018, https://hal.inria.fr/hal-01900327

[21] S. DUCASSE, H. ROCHA, S. BRAGAGNOLO, M. DENKER, C. FRANCOMME. *SmartAnvil: Open-Source Tool Suite for Smart Contract Analysis*, in "Blockchain and Web 3.0: Social, economic, and technological challenges", Routledge, February 2019, https://hal.inria.fr/hal-01940287

Research Reports

[22] J. DELPLANQUE, S. DUCASSE, A. BLACK, G. POLITO. *Rotten Green Tests A First Analysis*, Inria Lille Nord Europe - Laboratoire CRIStAL - Université de Lille ; Portland State University, Oregon, USA, June 2018, https://hal.archives-ouvertes.fr/hal-01819302

[23] M. DENKER, N. ANQUETIL, S. DUCASSE, A. ETIEN, D. POLLET. *Project-Team RMoD (Analyses and Language Constructs for Object-Oriented Application Evolution) 2017 Activity Report*, Inria Lille - Nord Europe, January 2018, https://hal.inria.fr/hal-01683649

References in notes

[24] N. ANQUETIL. *A Comparison of Graphs of Concept for Reverse Engineering*, in "Proceedings of the 8th International Workshop on Program Comprehension", Washington, DC, USA, IWPC'00, IEEE Computer Society, 2000, pp. 231–, http://rmod.lille.inria.fr/archives/papers/Anqu00b-ICSM-GraphsConcepts.pdf

[25] A. BERGEL, S. DUCASSE, O. NIERSTRASZ. *Classbox/J: Controlling the Scope of Change in Java*, in "Proceedings of 20th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05)", New York, NY, USA, ACM Press, 2005, pp. 177–189 [*DOI :* 10.1145/1094811.1094826], http://scg.unibe.ch/archive/papers/Berg05bclassboxjOOPSLA.pdf

[26] A. BERGEL, S. DUCASSE, O. NIERSTRASZ, R. WUYTS. *Stateful Traits*, in "Advances in Smalltalk — Proceedings of 14th International Smalltalk Conference (ISC 2006)", LNCS, Springer, August 2007, vol. 4406, pp. 66–90, http://dx.doi.org/10.1007/978-3-540-71836-9_3

[27] A. BERGEL, S. DUCASSE, O. NIERSTRASZ, R. WUYTS. *Stateful Traits and their Formalization*, in "Journal of Computer Languages, Systems and Structures", 2008, vol. 34, n° 2-3, pp. 83–108, http://dx.doi.org/10.1016/j.cl.2007.05.003

[28] A. P. BLACK, N. SCHÄRLI, S. DUCASSE. *Applying Traits to the Smalltalk Collection Hierarchy*, in "Proceedings of 17th International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'03)", October 2003, vol. 38, pp. 47–64 [*DOI :* 10.1145/949305.949311], http://scg.unibe.ch/archive/papers/Blac03aTraitsHierarchy.pdf

[29] G. BRACHA, D. UNGAR. *Mirrors: design principles for meta-level facilities of object-oriented programming languages*, in "Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04), ACM SIGPLAN Notices", New York, NY, USA, ACM Press, 2004, pp. 331–344, http://bracha.org/mirrors.pdf

[30] D. CAROMEL, J. VAYSSIÈRE. *Reflections on MOPs, Components, and Java Security*, in "ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming", Springer-Verlag, 2001, pp. 256–274

[31] D. CAROMEL, J. VAYSSIÈRE. *A security framework for reflective Java applications*, in "Software: Practice and Experience", 2003, vol. 33, n° 9, pp. 821–846, http://dx.doi.org/10.1002/spe.528

[32] P. COINTE. *Metaclasses are First Class: the ObjVlisp Model*, in "Proceedings OOPSLA '87, ACM SIGPLAN Notices", December 1987, vol. 22, pp. 156–167

[33] S. DENIER. *Traits Programming with AspectJ*, in "Actes de la Première Journée Francophone sur le Développement du Logiciel par Aspects (JFDLPA'04)", Paris, France, P. COINTE (editor), September 2004, pp. 62–78

[34] S. DUCASSE, T. GÎRBA. *Using Smalltalk as a Reflective Executable Meta-Language*, in "International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006)", Berlin, Germany, LNCS, Springer-Verlag, 2006, vol. 4199, pp. 604–618 [*DOI :* 10.1007/11880240_42], http://scg.unibe.ch/archive/papers/Duca06dMOOSEMODELS2006.pdf

[35] S. DUCASSE, T. GÎRBA, M. LANZA, S. DEMEYER. *Moose: a Collaborative and Extensible Reengineering Environment*, in "Tools for Software Maintenance and Reengineering", Milano, RCOST / Software Technology Series, Franco Angeli, 2005, pp. 55–71, http://scg.unibe.ch/archive/papers/Duca05aMooseBookChapter.pdf

[36] S. DUCASSE, O. NIERSTRASZ, N. SCHÄRLI, R. WUYTS, A. P. BLACK. *Traits: A Mechanism for fine-grained Reuse*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", March 2006, vol. 28, n° 2, pp. 331–388 [*DOI :* 10.1145/1119479.1119483], http://scg.unibe.ch/archive/papers/Duca06bTOPLASTraits.pdf

[37] S. DUCASSE, R. WUYTS, A. BERGEL, O. NIERSTRASZ. *User-Changeable Visibility: Resolving Unanticipated Name Clashes in Traits*, in "Proceedings of 22nd International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'07)", New York, NY, USA, ACM Press, October 2007, pp. 171–190 [*DOI :* 10.1145/1297027.1297040], http://scg.unibe.ch/archive/papers/Duca07b-FreezableTrait.pdf

[38] A. DUNSMORE, M. ROPER, M. WOOD. *Object-Oriented Inspection in the Face of Delocalisation*, in "Proceedings of ICSE '00 (22nd International Conference on Software Engineering)", ACM Press, 2000, pp. 467–476

[39] K. FISHER, J. REPPY. *Statically typed traits*, University of Chicago, Department of Computer Science, December 2003, n° TR-2003-13

[40] P. W. L. FONG, C. ZHANG. *Capabilities as alias control: Secure cooperation in dynamically extensible systems*, Department of Computer Science, University of Regina, 2004

[41] M. FURR, J.-H. AN, J. S. FOSTER. *Profile-guided static typing for dynamic scripting languages*, in "OOPSLA'09", 2009

[42] A. GOLDBERG. *Smalltalk 80: the Interactive Programming Environment*, Addison Wesley, Reading, Mass., 1984

[43] L. GONG. *New security architectural directions for Java*, in "compcon", 1997, vol. 0, 97 p. , http://dx.doi.org/10.1109/CMPCON.1997.584679

[44] M. HICKS, S. NETTLES. *Dynamic software updating*, in "ACM Transactions on Programming Languages and Systems", nov 2005, vol. 27, n° 6, pp. 1049–1096, http://dx.doi.org/10.1145/1108970.1108971

[45] G. KICZALES, J. DES RIVIÈRES, D. G. BOBROW. *The Art of the Metaobject Protocol*, MIT Press, 1991

[46] G. KICZALES, L. RODRIGUEZ. *Efficient Method Dispatch in PCL*, in "Proceedings of ACM conference on Lisp and Functional Programming", Nice, 1990, pp. 99–105

[47] R. KOSCHKE. *Atomic Architectural Component Recovery for Program Understanding and Evolution*, Universität Stuttgart, 2000, http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIS-2000-05&mod=0&engl=0&inst=PS

[48] S. LIANG, G. BRACHA. *Dynamic Class Loading in the Java Virtual Machine*, in "Proceedings of OOPSLA '98, ACM SIGPLAN Notices", 1998, pp. 36–44

[49] L. LIQUORI, A. SPIWACK. *FeatherTrait: A Modest Extension of Featherweight Java*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", 2008, vol. 30, n° 2, pp. 1–32 [*DOI : 10.1145/1330017.1330022*], http://www-sop.inria.fr/members/Luigi.Liquori/PAPERS/toplas-07.pdf

[50] B. LIVSHITS, T. ZIMMERMANN. *DynaMine: finding common error patterns by mining software revision histories*, in "SIGSOFT Software Engineering Notes", September 2005, vol. 30, n° 5, pp. 296-305

[51] R. C. MARTIN. *Agile Software Development. Principles, Patterns, and Practices*, Prentice-Hall, 2002

[52] M. S. MILLER. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*, Johns Hopkins University, Baltimore, Maryland, USA, May 2006

[53] M. S. MILLER, C. MORNINGSTAR, B. FRANTZ. *Capability-based Financial Instruments*, in "FC '00: Proceedings of the 4th International Conference on Financial Cryptography", Springer-Verlag, 2001, vol. 1962, pp. 349–378

[54] O. NIERSTRASZ, S. DUCASSE, N. SCHÄRLI. *Flattening Traits*, in "Journal of Object Technology", May 2006, vol. 5, n° 4, pp. 129–148, http://www.jot.fm/issues/issue_2006_05/article4

[55] P. J. QUITSLUND. *Java Traits — Improving Opportunities for Reuse*, OGI School of Science & Engineering, Beaverton, Oregon, USA, September 2004, n° CSE-04-005

[56] J. REPPY, A. TURON. *A Foundation for Trait-based Metaprogramming*, in "International Workshop on Foundations and Developments of Object-Oriented Languages", 2006

[57] F. RIVARD. *Pour un lien d'instanciation dynamique dans les langages à classes*, in "JFLA96", Inria — collection didactique, January 1996

[58] J. H. SALTZER, M. D. SCHOROEDER. *The Protection of Information in Computer Systems*, in "Fourth ACM Symposium on Operating System Principles", IEEE, September 1975, vol. 63, pp. 1278–1308

[59] N. SANGAL, E. JORDAN, V. SINHA, D. JACKSON. *Using Dependency Models to Manage Complex Software Architecture*, in "Proceedings of OOPSLA'05", 2005, pp. 167–176

[60] N. SCHÄRLI, A. P. BLACK, S. DUCASSE. *Object-oriented Encapsulation for Dynamically Typed Languages*, in "Proceedings of 18th International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'04)", October 2004, pp. 130–149 [*DOI :* 10.1145/1028976.1028988], http://scg.unibe.ch/archive/papers/Scha04bOOEncapsulation.pdf

[61] N. SCHÄRLI, S. DUCASSE, O. NIERSTRASZ, A. P. BLACK. *Traits: Composable Units of Behavior*, in "Proceedings of European Conference on Object-Oriented Programming (ECOOP'03)", LNCS, Springer Verlag, July 2003, vol. 2743, pp. 248–274 [*DOI :* 10.1007/B11832], http://scg.unibe.ch/archive/papers/Scha03aTraits.pdf

[62] C. SMITH, S. DROSSOPOULOU. *Chai: Typed Traits in Java*, in "Proceedings ECOOP 2005", 2005

[63] G. SNELTING, F. TIP. *Reengineering Class Hierarchies using Concept Analysis*, in "ACM Trans. Programming Languages and Systems", 1998

[64] K. J. SULLIVAN, W. G. GRISWOLD, Y. CAI, B. HALLEN. *The Structure and Value of Modularity in SOftware Design*, in "ESEC/FSE 2001", 2001

[65] D. VAINSENCHER. *MudPie: layers in the ball of mud*, in "Computer Languages, Systems & Structures", 2004, vol. 30, n^{o} 1-2, pp. 5–19

[66] N. WILDE, R. HUITT. *Maintenance Support for Object-Oriented Programs*, in "IEEE Transactions on Software Engineering", December 1992, vol. SE-18, n^{o} 12, pp. 1038–1044