IN PARTNERSHIP WITH:
CNRS

Université Pierre et Marie Curie (Paris 6)

Activity Report 2017

Project-Team WHISPER

Well Honed Infrastructure Software for Programming Environments and Runtimes

IN COLLABORATION WITH: Laboratoire d’informatique de Paris 6 (LIP6)

RESEARCH CENTER
Paris

THEME
Distributed Systems and middleware

Table of contents

1. **Personnel** 1
2. **Overall Objectives** 2
3. **Research Program** 3
   3.1. Scientific Foundations 3
      3.1.1. Program analysis 3
      3.1.2. Domain Specific Languages 4
         3.1.2.1. Traditional approach. 4
         3.1.2.2. Embedding DSLs. 4
         3.1.2.3. Certifying DSLs. 5
   3.2. Research direction: Tools for improving legacy infrastructure software 5
   3.3. Research direction: developing infrastructure software using Domain Specific Languages 6
4. **Application Domains** 6
   4.1. Linux 6
   4.2. Device Drivers 7
5. **Highlights of the Year** 7
6. **New Software and Platforms** 8
   6.1. Coccinelle 8
   6.2. Prequel 8
7. **New Results** 8
   7.1. Software engineering for infrastructure software 8
   7.2. Trustworthy domain-specific compilers 9
   7.3. Algebra of programming 10
   7.4. Developing infrastructure software using Domain Specific Languages 10
8. **Bilateral Contracts and Grants with Industry** 11
9. **Partnerships and Cooperations** 11
   9.1. Regional Initiatives 11
   9.2. National Initiatives 11
   9.3. International Initiatives 12
      9.3.1. Inria International Labs 12
      9.3.2. Inria International Partners 12
   9.4. International Research Visitors 12
         9.4.1.1. Internships 12
         9.4.1.2. Research Stays Abroad 12
10. **Dissemination** 12
   10.1. Promoting Scientific Activities 12
      10.1.1. Scientific Events Organisation 12
      10.1.2. Scientific Events Selection 13
         10.1.2.1. Chair of Conference Program Committees 13
         10.1.2.2. Member of the Conference Program Committees 13
      10.1.3. Journal 13
         10.1.3.1. Member of the Editorial Boards 13
         10.1.3.2. Reviewer - Reviewing Activities 13
      10.1.4. Invited Talks 13
      10.1.5. Research Administration 13
   10.2. Teaching - Supervision - Juries 13
      10.2.1. Teaching 13
      10.2.2. Supervision 14
      10.2.3. Juries 14
   10.3. Popularization 14

11. Bibliography **14**

Project-Team WHISPER

Creation of the Team: 2014 May 15, updated into Project-Team: 2015 December 01

Keywords:

Computer Science and Digital Science:

A1. - Architectures, systems and networks
A1.1.1. - Multicore, Manycore
A2. - Software
A2.1.6. - Concurrent programming
A2.1.10. - Domain-specific languages
A2.1.11. - Proof languages
A2.2.1. - Static analysis
A2.2.3. - Run-time systems
A2.3.1. - Embedded systems
A2.3.3. - Real-time systems
A2.4. - Verification, reliability, certification
A2.4.3. - Proofs
A2.5. - Software engineering
A2.6. - Infrastructure software
A2.6.1. - Operating systems
A2.6.2. - Middleware
A2.6.3. - Virtual machines

Other Research Topics and Application Domains:

B5. - Industry of the future
B5.2.1. - Road vehicles
B5.2.3. - Aviation
B5.2.4. - Aerospace
B6.1. - Software industry
B6.1.1. - Software engineering
B6.1.2. - Software evolution, maintenance
B6.3.3. - Network Management
B6.5. - Information systems
B6.6. - Embedded systems

1. Personnel

Research Scientists

Gilles Muller [Team leader, Inria, Senior Researcher, HDR]
Pierre-Évariste Dagand [CNRS, Researcher]
Julia Lawall [Inria, Senior Researcher]

Faculty Member

Bertil Folliot [Univ Pierre et Marie Curie, Professor, HDR]

Post-Doctoral Fellow
Van-Anh Nguyen [Univ Pierre et Marie Curie, financed by ANR ITrans]

PhD Students
Cédric Courtaud [Thales]
Redha Gouicem [Univ Pierre et Marie Curie]
Lucas Serrano [Univ Pierre et Marie Curie, from Nov 2017]
Darius Mercadier [Univ Pierre et Marie Curie, from Nov 2017]

Technical staff
Antoine Blin [Inria, granted by ORANGE SA]
Lucas Serrano [Inria, from Sep 2017 until Oct 2017]

Administrative Assistants
Nelly Maloisel [Inria, Assistant]
Eugène Kamdem [UPMC, Assistant]

Visiting Scientist
Gregory Kroah-Hartman [Linux Foundation, until Jul 2017]

2. Overall Objectives

2.1. Overall Objectives

The focus of Whisper is on how to develop (new) and improve (existing) infrastructure software. Infrastructure software (also called systems software) is the software that underlies all computing. Such software allows applications to access resources and provides essential services such as memory management, synchronization and inter-process interactions. Starting bottom-up from the hardware, examples include virtual machine hypervisors, operating systems, managed runtime environments, standard libraries, and browsers, which amount to the new operating system layer for Internet applications. For such software, efficiency and correctness are fundamental. Any overhead will impact the performance of all supported applications. Any failure will prevent the supported applications from running correctly. Since computing now pervades our society, with few paper backup solutions, correctness of software at all levels is critical. Formal methods are increasingly being applied to operating systems code in the research community [45], [51], [90]. Still, such efforts require a huge amount of manpower and a high degree of expertise which makes this work difficult to replicate in standard infrastructure-software development.

In terms of methodology, Whisper is at the interface of the domains of operating systems, software engineering and programming languages. Our approach is to combine the study of problems in the development of real-world infrastructure software with concepts in programming language design and implementation, *e.g.*, of domain-specific languages, and knowledge of low-level system behavior. A focus of our work is on providing support for legacy code, while taking the needs and competences of ordinary system developers into account.

We aim at providing solutions that can be easily learned and adopted by system developers in the short term. Such solutions can be tools, such as Coccinelle [1], [8], [9] for transforming C programs, or domain-specific languages such as Devil [7] and Bossa [6] for designing drivers and kernel schedulers. Due to the small size of the team, Whisper mainly targets operating system kernels and runtimes for programming languages. We put an emphasis on achieving measurable improvements in performance and safety in practice, and on feeding these improvements back to the infrastructure software developer community.

3. Research Program

3.1. Scientific Foundations

3.1.1. Program analysis

A fundamental goal of the research in the Whisper team is to elicit and exploit the knowledge found in existing code. To do this in a way that scales to a large code base, systematic methods are needed to infer code properties. We may build on either static [35], [37], [39] or dynamic analysis [59], [62], [69]. Static analysis consists of approximating the behavior of the source code from the source code alone, while dynamic analysis draws conclusions from observations of sample executions, typically of test cases. While dynamic analysis can be more accurate, because it has access to information about actual program behavior, obtaining adequate test cases is difficult. This difficulty is compounded for infrastructure software, where many, often obscure, cases must be handled, and external effects such as timing can have a significant impact. Thus, we expect to primarily use static analyses. Static analyses come in a range of flavors, varying in the extent to which the analysis is *sound*, *i.e.*, the extent to which the results are guaranteed to reflect possible run-time behaviors.

One form of sound static analysis is *abstract interpretation* [37]. In abstract interpretation, atomic terms are interpreted as sound abstractions of their values, and operators are interpreted as functions that soundly manipulate these abstract values. The analysis is then performed by interpreting the program in a compositional manner using these abstracted values and operators. Alternatively, *dataflow analysis* [50] iteratively infers connections between variable definitions and uses, in terms of local transition rules that describe how various kinds of program constructs may impact variable values. Schmidt has explored the relationship between abstract interpretation and dataflow analysis [77]. More recently, more general forms of symbolic execution [35] have emerged as a means of understanding complex code. In symbolic execution, concrete values are used when available, and these are complemented by constraints that are inferred from terms for which only partial information is available. Reasoning about these constraints is then used to prune infeasible paths, and obtain more precise results. A number of works apply symbolic execution to operating systems code [31], [33].

While sound approaches are guaranteed to give correct results, they typically do not scale to the very diverse code bases that are prevalent in infrastructure software. An important insight of Engler et al. [42] was that valuable information could be obtained even when sacrificing soundness, and that sacrificing soundness could make it possible to treat software at the scales of the kernels of the Linux or BSD operating systems. Indeed, for certain types of problems, on certain code bases, that may mostly follow certain coding conventions, it may mostly be safe to *e.g.*, ignore the effects of aliases, assume that variable values are unchanged by calls to unanalyzed functions, etc. Real code has to be understood by developers and thus cannot be too complicated, so such simplifying assumptions are likely to hold in practice. Nevertheless, approaches that sacrifice soundness also require the user to manually validate the results. Still, it is likely to be much more efficient for the user to perform a potentially complex manual analysis in a specific case, rather than to implement all possible required analyses and apply them everywhere in the code base. A refinement of unsound analysis is the CEGAR approach [36], in which a highly approximate analysis is complemented by a sound analysis that checks the individual reports of the approximate analysis, and then any errors in reasoning detected by the sound analysis are used to refine the approximate analysis. The CEGAR approach has been applied effectively on device driver code in tools developed at Microsoft [23]. The environment in which the driver executes, however, is still represented by possibly unsound approximations.

Going further in the direction of sacrificing soundness for scalability, the software engineering community has recently explored a number of approaches to code understanding based on techniques developed in the areas of natural language understanding, data mining, and information retrieval. These approaches view code, as well as other software-reated artifacts, such as documentation and postings on mailing lists, as bags of words structured in various ways. Statistical methods are then used to collect words or phrases that seem to be highly correlated, independently of the semantics of the program constructs that connect them. The obliviousness to program semantics can lead to many false positives (invalid conclusions) [55], but can also highlight trends that are not apparent at the low level of individual program statements. We have previously explored combining such statistical methods with more traditional static analysis in identifying faults in the usage of constants in Linux kernel code [54].

3.1.2. Domain Specific Languages

Writing low-level infrastructure code is tedious and difficult, and verifying it is even more so. To produce non-trivial programs, we could benefit from moving up the abstraction stack to enable both programming and proving as quickly as possible. Domain-specific languages (DSLs), also known as *little languages*, are a means to that end [5] [63].

3.1.2.1. Traditional approach.

Using little languages to aid in software development is a tried-and-trusted technique [80] by which programmers can express high-level ideas about the system at hand and avoid writing large quantities of formulaic C boilerplate.

This approach is typified by the Devil language for hardware access [7]. An OS programmer describes the register set of a hardware device in the high-level Devil language, which is then compiled into a library providing C functions to read and write values from the device registers. In doing so, Devil frees the programmer from having to write extensive bit-manipulation macros or inline functions to map between the values the OS code deals with, and the bit-representation used by the hardware: Devil generates code to do this automatically.

However, DSLs are not restricted to being “stub” compilers from declarative specifications. The Bossa language [6] is a prime example of a DSL involving imperative code (syntactically close to C) while offering a high-level of abstraction. This design of Bossa enables the developer to implement new process scheduling policies at a level of abstraction tailored to the application domain.

Conceptually, a DSL both abstracts away low-level details and justifies the abstraction by its semantics. In principle, it reduces development time by allowing the programmer to focus on high-level abstractions. The programmer needs to write less code, in a language with syntax and type checks adapted to the problem at hand, thus reducing the likelihood of errors.

3.1.2.2. Embedding DSLs.

The idea of a DSL has yet to realize its full potential in the OS community. Indeed, with the notable exception of interface definition languages for remote procedure call (RPC) stubs, most OS code is still written in a low-level language, such as C. Where DSL code generators are used in an OS, they tend to be extremely simple in both syntax and semantics. We conjecture that the effort to implement a given DSL usually outweighs its benefit. We identify several serious obstacles to using DSLs to build a modern OS: specifying what the generated code will look like, evolving the DSL over time, debugging generated code, implementing a bug-free code generator, and testing the DSL compiler.

Filet-o-Fish (FoF) [3] addresses these issues by providing a framework in which to build correct code generators from semantic specifications. This framework is presented as a Haskell library, enabling DSL writers to *embed* their languages within Haskell. DSL compilers built using FoF are quick to write, simple, and compact, but encode rigorous semantics for the generated code. They allow formal proofs of the run-time behavior of generated code, and automated testing of the code generator based on randomized inputs, providing greater test coverage than is usually feasible in a DSL. The use of FoF results in DSL compilers that OS developers can quickly implement and evolve, and that generate provably correct code. FoF has been used to build a number of domain-specific languages used in Barrelfish, [24] an OS for heterogeneous multicore systems developed at ETH Zurich.

The development of an embedded DSL requires a few supporting abstractions in the host programming language. FoF was developed in the purely functional language Haskell, thus benefiting from the type class mechanism for overloading, a flexible parser offering convenient syntactic sugar, and purity enabling a more algebraic approach based on small, composable combinators. Object-oriented languages – such as Smalltalk [44] and its descendant Pharo [28] – or multi-paradigm languages – such as the Scala programming language [66] – also offer a wide range of mechanisms enabling the development of embedded DSLs. Perhaps suprisingly, a low-level imperative language – such as C – can also be extended so as to enable the development of embedded compilers [25].

3.1.2.3. Certifying DSLs.

Whilst automated and interactive software verification tools are progressively being applied to larger and larger programs, we have not yet reached the point where large-scale, legacy software – such as the Linux kernel – could formally be proved "correct". DSLs enable a pragmatic approach, by which one could realistically strengthen a large legacy software by first narrowing down its critical component(s) and then focus our verification efforts onto these components.

Dependently-typed languages, such as Coq or Idris, offer an ideal environment for embedding DSLs [34], [29] in a unified framework enabling verification. Dependent types support the type-safe embedding of object languages and Coq's mixfix notation system enables reasonably idiomatic domain-specific concrete syntax. Coq's powerful abstraction facilities provide a flexible framework in which to not only implement and verify a range of domain-specific compilers [3], but also to combine them, and reason about their combination.

Working with many DSLs optimizes the "horizontal" compositionality of systems, and favors reuse of building blocks, by contrast with the "vertical" composition of the traditional compiler pipeline, involving a stack of comparatively large intermediate languages that are harder to reuse the higher one goes. The idea of building compilers from reusable building blocks is a common one, of course. But the interface contracts of such blocks tend to be complex, so combinations are hard to get right. We believe that being able to write and verify formal specifications for the pieces will make it possible to know when components can be combined, and should help in designing good interfaces.

Furthermore, the fact that Coq is also a system for formalizing mathematics enables one to establish a close, formal connection between embedded DSLs and non-trivial domain-specific models. The possibility of developing software in a truly "model-driven" way is an exciting one. Following this methodology, we have implemented a certified compiler from regular expressions to x86 machine code [4]. Interestingly, our development crucially relied on an existing Coq formalization, due to Braibant and Pous, [30] of the theory of Kleene algebras.

While these individual experiments seem to converge toward embedding domain-specific languages in rich type theories, further experimental validation is required. Indeed, Barrelfish is an extremely small software compared to the Linux kernel. The challenge lies in scaling this methodology up to large software systems. Doing so calls for a unified platform enabling the development of a myriad of DSLs, supporting code reuse across DSLs as well as providing support for mechanically-verified proofs.

3.2. Research direction: Tools for improving legacy infrastructure software

A cornerstone of our work on legacy infrastructure software is the Coccinelle program matching and transformation tool for C code. Coccinelle has been in continuous development since 2005. Today, Coccinelle is extensively used in the context of Linux kernel development, as well as in the development of other software, such as wine, python, kvm, and systemd. Currently, Coccinelle is a mature software project, and no research is being conducted on Coccinelle itself. Instead, we leverage Coccinelle in other research projects [26], [27], [67], [70], [74], [76], [79], [60],[16], both for code exploration, to better understand at a large scale problems in Linux development, and as an essential component in tools that require program matching and transformation. The continuing development and use of Coccinelle is also a source of visibility in the Linux kernel developer community. We submitted the first patches to the Linux kernel based on Coccinelle in 2007. Since then, over 5500 patches have been accepted into the Linux kernel based on the use of Coccinelle, including around 3000 by over 500 developers from outside our research group.

Our recent work has focused on driver porting. Specifically, we have considered the problem of porting a Linux device driver across versions, particularly backporting, in which a modern driver needs to be used by a client who, typically for reasons of stability, is not able to update their Linux kernel to the most recent version. When multiple drivers need to be backported, they typically need many common changes, suggesting that Coccinelle could be applicable. Using Coccinelle, however, requires writing backporting transformation rules. In order to more fully automate the backporting (or symmetrically forward porting) process, these rules should be generated automatically. We have carried out a preliminary study in this direction with David Lo of Singapore Management University; this work, published at ICSME 2016 [82], is limited to a port from one version to the next one, in the case where the amount of change required is limited to a single line of code. Whisper has been awarded an ANR PRCI grant to collaborate with the group of David Lo on scaling up the rule inference process and proposing a fully automatic porting solution.

3.3. Research direction: developing infrastructure software using Domain Specific Languages

We wish to pursue a *declarative* approach to developing infrastructure software. Indeed, there exists a significant gap between the high-level objectives of these systems and their implementation in low-level, imperative programming languages. To bridge that gap, we propose an approach based on domain-specific languages (DSLs). By abstracting away boilerplate code, DSLs increase the productivity of systems programmers. By providing a more declarative language, DSLs reduce the complexity of code, thus the likelihood of bugs.

Traditionally, systems are built by accretion of several, independent DSLs. For example, one might use Devil [7] to interact with devices, Bossa [6] to implement the scheduling policies. However, much effort is duplicated in implementing the back-ends of the individual DSLs. Our long term goal is to design a unified framework for developing and composing DSLs, following our work on Filet-o-Fish [3]. By providing a single conceptual framework, we hope to amortize the development cost of a myriad of DSLs through a principled approach to reusing and composing them.

Beyond the software engineering aspects, a unified platform brings us closer to the implementation of mechanically-verified DSLs. Dagand's recent work using the Coq proof assistant as an x86 macro-assembler [4] is a step in that direction, which belongs to a larger trend of hosting DSLs in dependent type theories [29], [34], [64]. A key benefit of those approaches is to provide – by construction – a formal, mechanized semantics to the DSLs thus developed. This semantics offers a foundation on which to base further verification efforts, whilst allowing interaction with non-verified code. We advocate a methodology based on incremental, piece-wise verification. Whilst building fully-certified systems from the top-down is a worthwhile endeavor [51], we wish to explore a bottom-up approach by which one focuses first and foremost on crucial subsystems and their associated properties.

Our current work on DSLs has two complementary goals: (i) the design of a unified framework for developing and composing DSLs, following our work on Filet-o-Fish, and (ii) the design of domain-specific languages for domains where there is a critical need for code correctness, and corresponding methodologies for proving properties of the run-time behavior of the system.

4. Application Domains

4.1. Linux

Linux is an open-source operating system that is used in settings ranging from embedded systems to supercomputers. The most recent release of the Linux kernel, v4.14, comprises over 16 million lines of code, and supports 30 different families of CPU architectures, around 50 file systems, and thousands of device drivers. Linux is also in a rapid stage of development, with new versions being released roughly every 2.5 months. Recent versions have each incorporated around 13,500 commits, from around 1500 developers. These developers have a wide range of expertise, with some providing hundreds of patches per release, while others have contributed only one. Overall, the Linux kernel is critical software, but software in which the quality of the developed source code is highly variable. These features, combined with the fact that the Linux community is open to contributions and to the use of tools, make the Linux kernel an attractive target for software researchers. Tools that result from research can be directly integrated into the development of real software, where it can have a high, visible impact.

Starting from the work of Engler et al. [41], numerous research tools have been applied to the Linux kernel, typically for finding bugs [39], [58], [71], [81] or for computing software metrics [47], [87]. In our work, we have studied generic C bugs in Linux code [9], bugs in function protocol usage [52], [53], issues related to the processing of bug reports [75] and crash dumps [46], and the problem of backporting [70], [82], illustrating the variety of issues that can be explored on this code base. Unique among research groups working in this area, we have furthermore developed numerous contacts in the Linux developer community. These contacts provide insights into the problems actually faced by developers and serve as a means of validating the practical relevance of our work.

4.2. Device Drivers

Device drivers are essential to modern computing, to provide applications with access, via the operating system, to physical devices such as keyboards, disks, networks, and cameras. Development of new computing paradigms, such as the internet of things, is hampered because device driver development is challenging and error-prone, requiring a high level of expertise in both the targeted OS and the specific device. Furthermore, implementing just one driver is often not sufficient; today's computing landscape is characterized by a number of OSes, *e.g.*, Linux, Windows, MacOS, BSD and many real time OSes, and each is found in a wide range of variants and versions. All of these factors make the development, porting, backporting, and maintenance of device drivers a critical problem for device manufacturers, industry that requires specific devices, and even for ordinary users.

The last fifteen years have seen a number of approaches directed towards easing device driver development. Réveillère, who was supervised by G. Muller, proposes Devil [7], a domain-specific language for describing the low-level interface of a device. Chipounov *et al.* propose RevNic, [33] a template-based approach for porting device drivers from one OS to another. Ryzhyk *et al.* propose Termite, [72], [73] an approach for synthesizing device driver code from a specification of an OS and a device. Currently, these approaches have been successfully applied to only a small number of toy drivers. Indeed, Kadav and Swift [49] observe that these approaches make assumptions that are not satisfied by many drivers; for example, the assumption that a driver involves little computation other than the direct interaction between the OS and the device. At the same time, a number of tools have been developed for finding bugs in driver code. These tools include SDV [23], Coverity [41], CP-Miner, [57] PR-Miner [58], and Coccinelle [8]. These approaches, however, focus on analyzing existing code, and do not provide guidelines on structuring drivers.

In summary, there is still a need for a methodology that first helps the developer understand the software architecture of drivers for commonly used operating systems, and then provides tools for the maintenance of existing drivers.

5. Highlights of the Year

5.1. Highlights of the Year

As part of a collaborative effort with Timothy Bourke, Lélio Brun, Marc Pouzet (Parkas team), Xavier Leroy (Gallium team), Lionel Rieg (Collège de France) and Pierre-Évariste Dagand, our work on a certified Lustre compiler was accepted at PLDI [13].

Julia Lawall was invited to present a talk as part of the Colloquium Jacques Morgenstern at Inria - Sophia Antipolis. The talk was entitled "Coccinelle: synergy between programming language research and the Linux kernel". A video of the presentation is available. [1]

The work of Julia Lawall on the Linux kernel was featured in the Linux Foundation's 2017 Linux Kernel Development Report. [2]

[1] https://www.canal-u.tv/video/inria/coccinelle_synergy_between_programming_language_research_and_the_linux_kernel.38185
[2] https://www.linuxfoundation.org/2017-linux-kernel-report-landing-page

6. New Software and Platforms

6.1. Coccinelle

KEYWORDS: Code quality - Evolution - Infrastructure software
FUNCTIONAL DESCRIPTION: Coccinelle is a tool for code search and transformation for C programs. It has been extensively used for bug finding and evolutions in Linux kernel code.

- Participants: Gilles Muller, Julia Lawall, Nicolas Palix, Rene Rydhof Hansen and Thierry Martinez
- Partners: LIP6 - IRILL
- Contact: Julia Lawall
- URL: http://coccinelle.lip6.fr

6.2. Prequel

KEYWORDS: Code search - Git
SCIENTIFIC DESCRIPTION: The commit history of a code base such as the Linux kernel is a gold mine of information on how evolutions should be made, how bugs should be fixed, etc. Nevertheless, the high volume of commits available and the rudimentary filtering tools provided mean that it is often necessary to wade through a lot of irrelevant information before finding example commits that can help with a specific software development problem. To address this issue, we propose Prequel (Patch Query Language), which brings the descriptive power of code matching to the problem of querying a commit history.
FUNCTIONAL DESCRIPTION: Prequel is a tool for searching for complex patterns in the commits of software managed using git.

- Participants: Gilles Muller and Julia Lawall
- Partners: LIP6 - IRILL
- Contact: Julia Lawall
- URL: http://prequel-pql.gforge.inria.fr/

7. New Results

7.1. Software engineering for infrastructure software

Work in 2017 on the Linux kernel has focused on the problem of kernel device driver porting and on kernel compilation as a validation mechanism in the presence of variability. We have also completed a study with researchers at Singapore Management University on the relationship between the code coverage of test cases and the number of post-release defects, focusing on a range of popular open-source projects. Finally, we have worked with researchers at the University of Frankfurt on the design of a transformation language targeting data representation changes.

Porting Linux device drivers to target more recent and older Linux kernel versions to compensate for the ever-changing kernel interface is a continual problem for Linux device driver developers. Acquiring information about interface changes is a necessary, but tedious and error prone, part of this task. To address these problems, we have proposed two tools, *Prequel* and *gcc-reduce*, to help the developer collect the needed information. Prequel provides language support for querying git commit histories, while gcc-reduce translates error messages produced by compiling a driver with a target kernel into appropriate Prequel queries. We have used our approach in porting 33 device driver files over up to 3 years of Linux kernel history, amounting to hundreds of thousands of commits. In these experiments, for 3/4 of the porting issues, our approach highlighted commits that enabled solving the porting task. For many porting issues, our approach retrieves relevant commits in 30 seconds or less. This work was published at USENIX ATC [16] and a related talk was presented at Linuxcon Europe. The Prequel tool and some of our experimental results are available at http://prequel-pql.gforge.inria.fr/. The complete tool suite is available at http://select-new.gforge.inria.fr/.

The Linux kernel is highly configurable, and thus, in principle, any line of code can be included or excluded from the compiled kernel based on configuration operations. Configurability complicates the task of a *kernel janitor*, who cleans up faults across the code base. A janitor may not be familiar with the configuration options that trigger compilation of a particular code line, leading him to believe that a fix has been compile-checked when this is not the case. We have proposed JMake, a mutation-based tool for signaling changed lines that are not subjected to the compiler. JMake shows that for most of the 12,000 file-modifying commits between Linux v4.3 and v4.4 the configuration chosen by the kernel allyesconfig option is sufficient, once the janitor chooses the correct architecture. For most commits, this check requires only 30 seconds or less. We furthermore characterize the situations in which changed code is not subjected to compilation in practice. This work was published at DSN [15] and a related talk was presented at Linuxcon Europe. JMake is available at http://jmake-release.gforge.inria.fr/.

Testing is a pivotal activity in ensuring the quality of software. Code coverage is a common metric used as a yardstick to measure the efficacy and adequacy of testing. However, does higher coverage actually lead to a decline in post-release bugs? Do files that have higher test coverage actually have fewer bug reports? The direct relationship between code coverage and actual bug reports has not yet been analysed via a comprehensive empirical study on real bugs. In an empirical study, we have examined these questions in the context of 100 large open-source Java software projects based on their actual reported bugs. Our results show that coverage has an insignificant correlation with the number of bugs that are found after the release of the software at the project level, and no such correlation at the file level. This work was done in collaboration with researchers at Singapore Management University and has been published in the IEEE Transactions on Reliability [12].

Data representation migration is a program transformation that involves changing the type of a particular data structure, and then updating all of the operations that somehow depend on that data structure according to the new type. Changing the data representation can provide benefits such as improving efficiency and improving the quality of the computed results. Performing such a transformation is challenging, because it requires applying data-type specific changes to code fragments that may be widely scattered throughout the source code, connected by dataflow dependencies. Refactoring systems are typically sensitive to dataflow dependencies, but are not programmable with respect to the features of particular data types. Existing program transformation languages provide the needed flexibility, but do not concisely support reasoning about dataflow dependencies.

To address the needs of data representation migration, we have proposed a new approach to program transformation that relies on a notion of semantic dependency: every transformation step propagates the transformation process onward to code that somehow depends on the transformed code. Our approach provides a declarative transformation-specification language, for expressing type-specific transformation rules. Our approach further provides scoped rules, a mechanism for guiding rule application, and tags, a device for simple program analysis within our framework, to enable more powerful program transformations. Evaluation of our prototype based on our approach, targeting C and C++ software, shows that it can improve program performance and the precision of the computed results, and that it scales to programs of up to 3700 lines. This work was done in collaboration with researchers at the University of Frankfurt and was published at PEPM [18].

7.2. Trustworthy domain-specific compilers

This year, we concluded the correctness proof of the compiler back-end of the Lustre [32] synchronous dataflow language. Synchronous dataflow languages are widely used for the design of embedded systems: they allow a high-level description of the system and naturally lend themselves to a hierarchical design. Developed in collaboration with members of the Parkas team of Inria Paris (Tim Bourke, Lélio Brun, Marc Pouzet), the Gallium team of Inria Paris (Xavier Leroy) and Collège de France (Lionel Rieg), this work formalizes the compilation of a synchronous data-flow language into an imperative sequential language, which is eventually translated to Cminor [56], one of CompCert's intermediate languages. The proof has been developed and verified in the Coq theorem prover. This project illustrates perfectly our methodology: the design of synchronous dataflow languages is first governed by semantic considerations (Kahn process networks and the synchrony hypothesis) that are then reifed into syntactic artefacts. The implementation of a certified compiler highlights this dependency on semantics, forcing us to give as crisp a semantics as possible for the proof effort to be manageable. This work was published in a national conference [19] as well as in an international conference [13], both on the topic of language design and implementation.

Expanding upon these ideas, Darius Mercadier started his PhD with us in October. We are currently developing a synchronous dataflow language targeting verified and high-performance implementations of bitsliced algorithms, with application to cryptographical algorithms [40]. Our preliminary results [22] are encouraging.

7.3. Algebra of programming

We have pursued our study of the algebraic structures of programming languages, from a syntactic as well as semantics perspective. Tackling the semantics aspect, Pierre-Évariste Dagand published a journal article introducing the theory of ornaments [11] to a general audience of functional programmers. Ornaments amount to a domain-specific language, usually described in type theory, for describing structure-preserving changes in algebraic datatypes. Such descriptions can be used to improve code reuse as well as ease of refactoring in functional languages. This work is part of a wider effort by our community to foster the adoption of ornaments when programming with algebraic datatypes, be it in type theory [48] or general-purpose functional programming languages [65], [89]. Tackling the syntactic aspect and in collaboration with researchers at the University of Utrecht (Victor Miraldo, Wouter Swierstra), Pierre-Évariste Dagand has worked on a formalization of `diffs` for structured data [20]. This preliminary and foundational work aims at providing a typed specification to the problem of computing the difference of two pieces of structured data. Unlike previous approaches [43], following a type-theoretical approach allowed us to formalize the difference of two structure as a typed object. The task of computing the difference of two structured objects is then able to exploit this typing information to control the search space (which is otherwise gigantic). Having a typed difference also ensures that applying such a `diff` to a well-structured data results in either a failure (the difference is in conflict with the given file) or another well-structured data.

7.4. Developing infrastructure software using Domain Specific Languages

In terms of DSL design for domains where correctness is critical, our current focus is first on process scheduling for multicore architecture, and second on selfishness in distributed systems. Ten years ago, we developed Bossa, targeting process scheduling on unicore processors, and primarily focusing on the correctness of a scheduling policy with respect to the requirements of the target kernel. At that time, the main use cases were soft real-time applications, such as video playback. Bossa was and still continues to be used in teaching, because the associated verifications allow a student to develop a kernel-level process scheduling policy without the risk of a kernel crash. Today, however, there is again a need for the development of new scheduling policies, now targeting multicore architectures. As identified by Lozi *et al.* [61], large-scale server applications, having specific resource access properties, can exhibit pathological properties when run with the Linux kernel's various load balancing heuristics. We are working on a new domain-specific language, Ipanema, to enable verification of critical scheduling properties such as liveness and work-conservation; for the latter, we are exploring the use of the Leon theorem prover from EPFL [17]. A first version of the language has been designed and we expect to release a prototype of Ipanema working next year. The work around Ipanema is the subject of a very active collaboration between researchers at four institutions (Inria, University of Nice, University of Grenoble, and EPFL (groups of V. Kuncak and W. Zwaenepoel)). Baptiste Lepers (EPFL) is supported in 2017 as a postdoc as part of the Inria-EPFL joint laboratory.

Selfishness is one of the key problems that confronts developers of cooperative distributed systems (e.g., file-sharing networks, voluntary computing). It has the potential to severely degrade system performance and to lead to instability and failures. Current techniques for understanding the impact of selfish behaviours and designing effective countermeasures remain manual and time-consuming, requiring multi-domain expertise. To overcome these difficulties, we have proposed SEINE, a simulation framework for rapid modelling and evaluation of selfish behaviours in a cooperative system. SEINE relies on a domain-specific language (SEINE-L) for specifying selfishness scenarios, and provides semi-automatic support for their implementation and study in a state-of-the-art simulator. We show in a paper published at DSN 2017 [14] that (1) SEINE-L is expressive enough to specify fifteen selfishness scenarios taken from the literature, (2) SEINE is accurate in predicting the impact of selfishness compared to real experiments, and (3) SEINE substantially reduces the development effort compared to traditional manual approaches.

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Contracts with Industry

- Orange Labs, 2016-2018, 120 000 euros. The purpose of this contract is to apply the techniques developed in the context of the PhD of Antoine Blin to the domain of Software Defined Networks where network functions are run using virtual machines on commodity multicore machines.
- Thales Research, 2016-2018, 45 000 euros. The purpose of this contract is to enable the usage of multicore architectures in avionics systems. More precisely, our goal is to develop optimizations for a software TDMA hypervisor developed by Thales that provides full time-isolation of tasks. The PhD of Cédric Courtaud is supported by a CIFRE fellowship with Thales Research.
- OSADL, 2016-2017, development of the Prequel patch query language, 20 000 euros. OSADL is an organization headquartered in Germany that promotes and supports the use of open source software in the automation and machine industry. The project is in the context of the OSADL project SIL2LinuxMP bringing together various companies in automotive and embedded sytems with the goal of developing methodologies for certifying the basic components of a GNU/Linux-based RTOS.

9. Partnerships and Cooperations

9.1. Regional Initiatives

- City of Paris, 2016-2019, 100 000 euros. As part of the “Émergence - young team” program the city of Paris is supporting part of our work on domain-specific languages.

9.2. National Initiatives

9.2.1. ANR

ITrans - awarded in 2016, duration 2017 - 2020

Members: LIP6 (Whisper), David Lo (Singapore Management University)

Coordinator: Julia Lawall

Whisper members: Julia Lawall, Gilles Muller, Lucas Serrano, Van-Anh Nguyen

Funding: ANR PRCI, 287,820 euros.

Objectives:

Large, real-world software must continually change, to keep up with evolving requirements, fix bugs, and improve performance, maintainability, and security. This rate of change can pose difficulties for clients, whose code cannot always evolve at the same rate. This project will target the problems of *forward porting*, where one software component has to catch up to a code base with which it needs to interact, and *back porting*, in which it is desired to use a more modern component in a context where it is necessary to continue to use a legacy code base, focusing on the context of Linux device drivers. In this project, we will take a *history-guided source-code transformation-based* approach, which automatically traverses the history of the changes made to a software system, to find where changes in the code to be ported are required, gathers examples of the required changes, and generates change rules to incrementally back port or forward port the code. Our approach will be a success if it is able to automatically back and forward port a large number of drivers for the Linux operating system to various earlier and later versions of the Linux kernel with high accuracy while requiring minimal developer effort. This objective is not achievable by existing techniques.

9.3. International Initiatives

9.3.1. Inria International Labs

- EPFL-Inria Lab Our work on the Ipanema DSL [17] is done as part of the EPFL-Inria Lab. Baptiste Lepers (EPFL) is supported in 2017 as a joint postdoc between the Whisper and the groups of V. Kuncak and W. Zwaenepoel.

9.3.2. Inria International Partners

9.3.2.1. Informal International Partners

- We collaborate with David Lo and Lingxiao Jiang of Singapore Management University, who are experts in software mining, clone detection, and information retrieval techniques. Our work with Lo and/or Jiang has led to 8 joint publications since 2013 [12], [68], [78], [83], [84], [85], [88], [86], at conferences including ASE and ICSME. The ITrans ANR is a joint project with them.
- We collaborate with Christoph Reichenbach of the University of Lund and Krishna Narasimhan of Itemis (Germany) on program transformation [18] and the design of tools for code clone management.
- We collaborate with Wouter Swierstra of the University of Utrecht (Netherlands) on type-directed structured differences [20].
- We collaborate with Eric Tanter of the University of Chile (Chile) on the theoretical and practical aspects of dependent interoperability [38] in type theory.

9.4. International Research Visitors

9.4.1. Visits of International Scientists

As part of the Invited Professor program of LIP6, we have hosted Prof. Éric Tanter (University of Chile) for two weeks (December 2017) who took this opportunity to give an introductory master class as well as a research seminar on the topic of gradual typing.

9.4.1.1. Internships

- Lukas Gnirke, Oberlin College, January 2017, evaluation of our methodology for searching for examples to guide driver porting [16].
- Adina Johnson, Oberlin College, May - August 2017, analysis of the differences between the Linux kernel and the Android kernel.
- Jonathan Carroll, Oberlin College, May - August 2017, use of machine learning to identify stable-kernel relevant patches.
- Bhumika Goyal, October - November 2017, constification of Linux kernel structures, supported by the Linux Foundation's Core Infrastructure Initiative.
- Peio Borthelle, École Normale Supérieure de Lyon, June - July 2017, solving the Oware on a single machine.
- Darius Mercadier, Université Pierre et Marie Curie, January - August 2017, designing and implementing Usuba, a bitslicing compiler.

9.4.1.2. Research Stays Abroad

- Julia Lawall, visit to David Lo and Lingxiao Jiang at Singapore Management University (two weeks in May 2017).

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Organisation

10.1.1.1. Member of the Organizing Committees

- Gilles Muller: PLOS 2017

10.1.2. Scientific Events Selection

10.1.2.1. Chair of Conference Program Committees

- Julia Lawall: PLOS 2017

10.1.2.2. Member of the Conference Program Committees

- Gilles Muller: Usenix ATC 2017, DSN 2017, Systor 2017,
- Julia Lawall: ICSE NIER, ML workshop, OCaml workshop, SPLASH workshops.
- Pierre-Évariste Dagand: HOPE workshop

10.1.3. Journal

10.1.3.1. Member of the Editorial Boards

- Julia Lawall: Editorial board of Science of Computer Programing (2008 - present).

10.1.3.2. Reviewer - Reviewing Activities

- Gilles Muller: IEEE Transactions on Computer Systems
- Julia Lawall: Computer Languages, Empirical Software Engineering, IEEE Transactions on Reliability
- Pierre-Évariste Dagand: Journal of Functional Programming, Journal of Logical and Algebraic Methods in Programming, Journée Francophones des Langages Applicatifs

10.1.4. Invited Talks

- Gilles Muller: University of North Carolina, Inria Grenoble, University of Bordeaux.
- Julia Lawall: Colloquium Jacques Morgenstern Sopha Antipolis, Vrije Universiteit Brussel, University of Copenhagen, 2017 FSD Meeting.
- Pierre-Évariste Dagand: École Normale Supérieure de Cachan

10.1.5. Research Administration

- Pierre-Évariste Dagand: Member of the steering committee for the Colloquium d'Informatique de L'UPMC Sorbonne Universités, organizer of the Colloquium of Philippa Gardner (March 2017) and Timothy Roscoe (November 2017).
- Julia Lawall: IFIP TC secretary (2012 - present). Elected member of IFIP WG 2.11.

  Hiring committees: UPMC (PR)

  Board member of Software Heritage (https://www.softwareheritage.org/).

  Organized the Colloquium d'Informatique de L'UPMC Sorbonne Universités of Simon Peyton Jones (May 2017)
- Gilles Muller: EuroSys steering committee (2013-2017), elected member of IFIP WG 10.4 (Dependability), representative of Inria in Sorbonne University's advisory committee for research, member of the project committee board of the Inria Paris Center, member of the Paris committee for allocating post-docs, PhD stipends and sabbaticals.

  Hiring committees: University of Grenoble (MdC).
- Bertil Folliot: Elected member of the IFIP WG10.3 working group (Concurrent systems)

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

- Professional Licence: Bertil Folliot, Programmation C, L2, UPMC, France
- Professional Licence: Bertil Folliot, Lab projects, L2, UPMC, France

- Licence: Pierre-Évariste Dagand, Distributed cooperating objects, L3, UPMC, France
- Master: Pierre-Évariste Dagand, Specification and Validation of Programs, M2, UPMC, France
- Licence: Pierre-Évariste Dagand, INF311: Introduction to Programming, L1, École Polytechnique, France

10.2.2. Supervision

- PhD in progress : Mariem Saeid, soutenance en 2018, Jens Gustedt (Camus), Gilles Muller.
- PhD in progress : Cédric Courtaud, CIFRE Thalès, 2016-2019, Gilles Muller, Julien Sopéna (Regal).
- PhD in progress : Redha Gouicem, 2016-2019, Gilles Muller, Julien Sopéna (Regal).
- PhD in progress : Darius Mercadier, 2017-2020, Pierre-Évariste Dagand, Gilles Muller.
- PhD in progress : Lucas Serrano, 2017-2020, Julia Lawall.

10.2.3. Juries

- Gilles Muller: Reporter of the HDR of A. Tchana (U. Toulouse), member of the HDR of S. Ben Mokhtar (U. of Lyon).
- Julia Lawall: Vinh Tao (PhD, UPMC, president), Marcelino Cancio Rodriguez (PhD, University of Rennes, reporter), Victor Allombert (PhD, Paris Est-Creteil, reporter), Reinout Stevens (PhD, VUB, reporter).

10.3. Popularization

- Julia Lawall: Coordinator of the Outreachy internship program for the Linux kernel. Outreachy provides remote 3-month internships twice a year for women and other underrepresented minorities on open source projects. Julia Lawall also mentored Bhumika Goyal as part of this program.
- Julia Lawall, "Fast and Precise Retrieval of Forward and Back Porting Information for Linux Device Drivers", Open Source Summit Europe, October 2017.
- Julia Lawall, "JMake: Dependable Compilation for Kernel Janitors", Open Source Summit Europe, October 2017.
- Julia Lawall, "Panel Discussion: Outreachy Kernel Internship Report" (moderator), Open Source Summit Europe, October 2017.
- Julia Lawall, "Overview of Coccinelle", Linux Lund Conference, May 2018.
- Julia Lawall, "Constification of Linux kernel structures", OSADL Networking Day, May 2017.

11. Bibliography

Major publications by the team in recent years

[1] J. Brunel, D. Doligez, R. R. Hansen, J. L. Lawall, G. Muller. *A foundation for flow-based program matching using temporal logic and model checking*, in "POPL", Savannah, GA, USA, ACM, January 2009, pp. 114–126

[2] L. Burgy, L. Réveillère, J. L. Lawall, G. Muller. *Zebu: A Language-Based Approach for Network Protocol Message Processing*, in "IEEE Trans. Software Eng.", 2011, vol. 37, n^{o} 4, pp. 575-591

[3] P.-É. Dagand, A. Baumann, T. Roscoe. *Filet-o-Fish: practical and dependable domain-specific languages for OS development*, in "Programming Languages and Operating Systems (PLOS)", 2009, pp. 51–55

[4] A. KENNEDY, N. BENTON, J. B. JENSEN, P.-É. DAGAND. *Coq: The World's Best Macro Assembler?*, in "PPDP", Madrid, Spain, ACM, 2013, pp. 13–24

[5] G. MULLER, C. CONSEL, R. MARLET, L. P. BARRETO, F. MÉRILLON, L. RÉVEILLÈRE. *Towards Robust OSes for Appliances: A New Approach Based on Domain-specific Languages*, in "Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System", Kolding, Denmark, 2000, pp. 19–24

[6] G. MULLER, J. L. LAWALL, H. DUCHESNE. *A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation*, in "HASE - High Assurance Systems Engineering Conference", Heidelberg, Germany, IEEE, October 2005, pp. 56–65

[7] F. MÉRILLON, L. RÉVEILLÈRE, C. CONSEL, R. MARLET, G. MULLER. *Devil: An IDL for hardware programming*, in "Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI)", San Diego, California, USENIX Association, October 2000, pp. 17–30

[8] Y. PADIOLEAU, J. L. LAWALL, R. R. HANSEN, G. MULLER. *Documenting and Automating Collateral Evolutions in Linux Device Drivers*, in "EuroSys", Glasgow, Scotland, March 2008, pp. 247–260

[9] N. PALIX, G. THOMAS, S. SAHA, C. CALVÈS, J. L. LAWALL, G. MULLER. *Faults in Linux 2.6*, in "ACM Transactions on Computer Systems", June 2014, vol. 32, n^{o} 2, pp. 4:1–4:40

Publications of the year

Doctoral Dissertations and Habilitation Theses

[10] A. BLIN. *Towards an efficient use of multi-core processors in mixed criticality embedded systems*, Université Pierre et Marie Curie - Paris VI, January 2017, https://tel.archives-ouvertes.fr/tel-01624259

Articles in International Peer-Reviewed Journals

[11] P.-E. DAGAND. *The essence of ornaments*, in "Journal of Functional Programming", January 2017, vol. 27 [*DOI :* 10.1017/S0956796816000356], https://hal.archives-ouvertes.fr/hal-01461209

[12] P. SINGH KOCHHAR, D. LO, J. LAWALL, N. NAGAPPAN. *Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects*, in "IEEE Transactions on Reliability", December 2017, vol. 66, n^{o} 4, pp. 1213 - 1228 [*DOI :* 10.1109/TR.2017.2727062], https://hal.inria.fr/hal-01653728

International Conferences with Proceedings

[13] T. BOURKE, L. BRUN, P.-E. DAGAND, X. LEROY, M. POUZET, L. RIEG. *A Formally Verified Compiler for Lustre*, in "PLDI 2017 - 38th ACM SIGPLAN Conference on Programming Language Design and Implementation", Barcelone, Spain, ACM, June 2017, https://hal.inria.fr/hal-01512286

[14] G. L. COTA, S. BEN MOKHTAR, G. GIANINI, E. DAMIANI, J. LAWALL, G. MULLER, L. BRUNIE. *Analysing Selfishness Flooding with SEINE*, in "The 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17)", Denver, Colorado, United States, June 2017, pp. 603 - 614 [*DOI :* 10.1109/DSN.2017.51], https://hal.archives-ouvertes.fr/hal-01581628

[15] J. LAWALL, G. MULLER. *JMake: Dependable Compilation for Kernel Janitors*, in "The 47th IEEE/IFIP International Conference on Dependable Systems and Networks", Denver,Colorado, United States, IEEE/IFIP, June 2017 [*DOI :* 10.1109/DSN.2017.62], https://hal.inria.fr/hal-01555711

[16] J. LAWALL, D. PALINSKI, L. GNIRKE, G. MULLER. *Fast and Precise Retrieval of Forward and Back Porting Information for Linux Device Drivers*, in "2017 USENIX Annual Technical Conference", Santa Clara, CA, United States, July 2017, 12 p. , https://hal.inria.fr/hal-01556589

[17] B. LEPERS, W. ZWAENEPOEL, J.-P. LOZI, N. PALIX, R. GOUICEM, J. SOPENA, J. LAWALL, G. MULLER. *Towards Proving Optimistic Multicore Schedulers*, in "HotOS 2017 - 16th Workshop on Hot Topics in Operating Systems", Whistler, British Columbia, Canada, ACM SIGOPS, May 2017, 6 p. [*DOI :* 10.1145/3102980.3102984], https://hal.inria.fr/hal-01556597

[18] K. NARASIMHAN, C. REICHENBACH, J. LAWALL. *Interactive Data Representation Migration: Exploiting Program Dependence to Aid Program Transformation*, in "PEPM 2017 Workshop on Partial Evaluation and Program Manipulation", Paris, France, January 2017, https://hal.inria.fr/hal-01408266

National Conferences with Proceedings

[19] T. BOURKE, P.-E. DAGAND, M. POUZET, L. RIEG. *Vérification de la génération modulaire du code impératif pour Lustre*, in "JFLA 2017 - Vingt-huitième Journées Francophones des Langages Applicatifs", Gourette, France, January 2017, https://hal.inria.fr/hal-01403830

Conferences without Proceedings

[20] V. C. MIRALDO, P.-E. DAGAND, W. SWIERSTRA. *Type-directed diffing of structured data*, in "TyDe 2017 Proceedings of the 2nd ACM SIGPLAN International Workshop on Type-Driven Development", Oxford, United Kingdom, September 2017 [*DOI :* 10.1145/3122975.3122976], https://hal.archives-ouvertes.fr/hal-01673541

Other Publications

[21] P.-E. DAGAND, N. TABAREAU, É. TANTER. *Foundations of Dependent Interoperability*, December 2017, working paper or preprint, https://hal.inria.fr/hal-01629909

[22] D. MERCADIER, P.-É. DAGAND, L. LACASSAGNE, G. MULLER. *Usuba, Optimizing & Trustworthy Bitslicing Compiler*, December 2017, working paper or preprint, https://hal.archives-ouvertes.fr/hal-01657259

References in notes

[23] T. BALL, E. BOUNIMOVA, B. COOK, V. LEVIN, J. LICHTENBERG, C. MCGARVEY, B. ONDRUSEK, S. K. RAJAMANI, A. USTUNER. *Thorough Static Analysis of Device Drivers*, in "EuroSys", 2006, pp. 73–85

[24] A. BAUMANN, P. BARHAM, P.-É. DAGAND, T. HARRIS, R. ISAACS, S. PETER, T. ROSCOE, A. SCHÜPBACH, A. SINGHANIA. *The multikernel: A new OS architecture for scalable multicore systems*, in "SOSP", 2009, pp. 29–44

[25] T. F. BISSYANDÉ, L. RÉVEILLÈRE, J. L. LAWALL, Y.-D. BROMBERG, G. MULLER. *Implementing an embedded compiler using program transformation rules*, in "Software: Practice and Experience", 2013

[26] T. F. BISSYANDÉ, L. RÉVEILLÈRE, J. LAWALL, Y.-D. BROMBERG, G. MULLER. *Implementing an Embedded Compiler using Program Transformation Rules*, in "Software: Practice and Experience", February 2015, vol. 45, n^o^ 2, pp. 177-196, https://hal.archives-ouvertes.fr/hal-00844536

[27] T. F. BISSYANDÉ, L. RÉVEILLÈRE, J. LAWALL, G. MULLER. *Ahead of Time Static Analysis for Automatic Generation of Debugging Interfaces to the Linux Kernel*, in "Automated Software Engineering", May 2014, pp. 1-39 [*DOI :* 10.1007/s10515-014-0152-4], https://hal.archives-ouvertes.fr/hal-00992283

[28] A. P. BLACK, S. DUCASSE, O. NIERSTRASZ, D. POLLET. *Pharo by Example*, Square Bracket Associates, 2010

[29] E. BRADY, K. HAMMOND. *Resource-Safe Systems Programming with Embedded Domain Specific Languages*, in "14th International Symposium on Practical Aspects of Declarative Languages (PADL)", LNCS, Springer, 2012, vol. 7149, pp. 242–257

[30] T. BRAIBANT, D. POUS. *An Efficient Coq Tactic for Deciding Kleene Algebras*, in "1st International Conference on Interactive Theorem Proving (ITP)", LNCS, Springer, 2010, vol. 6172, pp. 163–178

[31] C. CADAR, D. DUNBAR, D. R. ENGLER. *KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs*, in "OSDI", 2008, pp. 209–224

[32] P. CASPI, N. HALBWACHS, D. PILAUD, J. PLAICE. *Lustre: a declarative language for programming synchronous systems*, in "14th ACM Symposium on Principles of Programming Languages", ACM, 1987

[33] V. CHIPOUNOV, G. CANDEA. *Reverse Engineering of Binary Device Drivers with RevNIC*, in "EuroSys", 2010, pp. 167–180

[34] A. CHLIPALA. *The Bedrock Structured Programming System: Combining Generative Metaprogramming and Hoare Logic in an Extensible Program Verifier*, in "ICFP", 2013, pp. 391–402

[35] L. A. CLARKE. *A system to generate test data and symbolically execute programs*, in "IEEE Transactions on Software Engineering", 1976, vol. 2, n^o^ 3, pp. 215–222

[36] E. CLARKE, O. GRUMBERG, S. JHA, Y. LU, H. VEITH. *Counterexample-guided abstraction refinement for symbolic model checking*, in "J. ACM", 2003, vol. 50, n^o^ 5, pp. 752–794

[37] P. COUSOT, R. COUSOT. *Abstract Interpretation: Past, Present and Future*, in "CSL-LICS", 2014, pp. 2:1–2:10

[38] P.-E. DAGAND, N. TABAREAU, É. TANTER. *Partial Type Equivalences for Verified Dependent Interoperability*, in "ICFP 2016 - 21st ACM SIGPLAN International Conference on Functional Programming", Nara, Japan, September 2016, pp. 298-310, http://dx.doi.org/10.1145/2951913.2951933

[39] I. DILLIG, T. DILLIG, A. AIKEN. *Sound, complete and scalable path-sensitive analysis*, in "PLDI", June 2008, pp. 270–280

[40] D. DINU, Y. L. CORRE, D. KHOVRATOVICH, L. PERRIN, J. GROSSSCHÄDL, A. BIRYUKOV. *Triathlon of Lightweight Block Ciphers for the Internet of Things*, 2015, Cryptology ePrint Archive, Report 2015/209

[41] D. R. ENGLER, B. CHELF, A. CHOU, S. HALLEM. *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, in "OSDI", 2000, pp. 1–16

[42] D. R. ENGLER, D. Y. CHEN, A. CHOU, B. CHELF. *Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code*, in "SOSP", 2001, pp. 57–72

[43] J. FALLERI, F. MORANDAT, X. BLANC, M. MARTINEZ, M. MONPERRUS. *Fine-grained and accurate source code differencing*, in "ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014", 2014, pp. 313–324, http://dx.doi.org/10.1145/2642937.2642982

[44] A. GOLDBERG, D. ROBSON. *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, 1983

[45] L. GU, A. VAYNBERG, B. FORD, Z. SHAO, D. COSTANZO. *CertiKOS: A Certified Kernel for Secure Cloud Computing*, in "Proceedings of the Second Asia-Pacific Workshop on Systems (APSys)", 2011, pp. 3:1–3:5

[46] L. GUO, J. L. LAWALL, G. MULLER. *Oops! Where did that code snippet come from?*, in "11th Working Conference on Mining Software Repositories, MSR", Hyderabad, India, ACM, May 2014, pp. 52–61

[47] A. ISRAELI, D. G. FEITELSON. *The Linux kernel as a case study in software evolution*, in "Journal of Systems and Software", 2010, vol. 83, n^{o} 3, pp. 485–501

[48] H.-S. KO, J. GIBBONS. *Programming with ornaments*, in "Journal of Functional Programming", 2017, vol. 27, http://dx.doi.org/10.1017/S0956796816000307

[49] A. KADAV, M. M. SWIFT. *Understanding modern device drivers*, in "ASPLOS", 2012, pp. 87–98

[50] G. A. KILDALL. *A Unified Approach to Global Program Optimization*, in "POPL", 1973, pp. 194–206

[51] G. KLEIN, K. ELPHINSTONE, G. HEISER, J. ANDRONICK, D. COCK, P. DERRIN, D. ELKADUWE, K. ENGELHARDT, R. KOLANSKI, M. NORRISH, T. SEWELL, H. TUCH, S. WINWOOD. *seL4: formal verification of an OS kernel*, in "SOSP", 2009, pp. 207–220

[52] J. L. LAWALL, J. BRUNEL, N. PALIX, R. R. HANSEN, H. STUART, G. MULLER. *WYSIWIB: Exploiting fine-grained program structure in a scriptable API-usage protocol-finding process*, in "Software, Practice Experience", 2013, vol. 43, n^{o} 1, pp. 67–92

[53] J. L. LAWALL, B. LAURIE, R. R. HANSEN, N. PALIX, G. MULLER. *Finding Error Handling Bugs in OpenSSL using Coccinelle*, in "Proceeding of the 8th European Dependable Computing Conference (EDCC)", Valencia, Spain, April 2010, pp. 191–196

[54] J. L. LAWALL, D. LO. *An automated approach for finding variable-constant pairing bugs*, in "25th IEEE/ACM International Conference on Automated Software Engineering", Antwerp, Belgium, September 2010, pp. 103–112

[55] C. LE GOUES, W. WEIMER. *Specification Mining with Few False Positives*, in "TACAS", York, UK, Lecture Notes in Computer Science, March 2009, vol. 5505, pp. 292–306

[56] X. LEROY. *Formal verification of a realistic compiler*, in "Communications of the ACM", 2009, vol. 52, n^{o} 7, pp. 107–115

[57] Z. LI, S. LU, S. MYAGMAR, Y. ZHOU. *CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code*, in "OSDI", 2004, pp. 289–302

[58] Z. LI, Y. ZHOU. *PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code*, in "Proceedings of the 10th European Software Engineering Conference", 2005, pp. 306–315

[59] D. LO, S. KHOO. *SMArTIC: towards building an accurate, robust and scalable specification miner*, in "FSE", 2006, pp. 265–275

[60] J.-P. LOZI, F. DAVID, G. THOMAS, J. LAWALL, G. MULLER. *Fast and Portable Locking for Multicore Architectures*, in "ACM Transactions on Computer Systems", January 2016 [*DOI :* 10.1145/2845079], https://hal.inria.fr/hal-01252167

[61] J. LOZI, B. LEPERS, J. R. FUNSTON, F. GAUD, V. QUÉMA, A. FEDOROVA. *The Linux scheduler: a decade of wasted cores*, in "Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, April 18-21, 2016", C. CADAR, P. PIETZUCH, K. KEETON, R. RODRIGUES (editors), ACM, 2016, pp. 1:1–1:16, http://doi.acm.org/10.1145/2901318.2901326

[62] S. LU, S. PARK, Y. ZHOU. *Finding Atomicity-Violation Bugs through Unserializable Interleaving Testing*, in "IEEE Transactions on Software Engineering", 2012, vol. 38, n^{o} 4, pp. 844–860

[63] M. MERNIK, J. HEERING, A. M. SLOANE. *When and How to Develop Domain-specific Languages*, in "ACM Comput. Surv.", December 2005, vol. 37, n^{o} 4, pp. 316–344, http://dx.doi.org/10.1145/1118890.1118892

[64] G. MORRISETT, G. TAN, J. TASSAROTTI, J.-B. TRISTAN, E. GAN. *RockSalt: better, faster, stronger SFI for the x86*, in "PLDI", 2012, pp. 395-404

[65] S. NAJD, S. P. JONES. *Trees that Grow*, in "Journal of Universal Computer Science", jan 2017, vol. 23, n^{o} 1, pp. 42–62

[66] M. ODERSKY, T. ROMPF. *Unifying functional and object-oriented programming with Scala*, in "Commun. ACM", 2014, vol. 57, n^{o} 4, pp. 76–86

[67] M. C. OLESEN, R. R. HANSEN, J. L. LAWALL, N. PALIX. *Coccinelle: Tool support for automated CERT C Secure Coding Standard certification*, in "Science of Computer Programming", October 2014, vol. 91, n^{o} B, pp. 141–160, https://hal.inria.fr/hal-01096185

[68] K. PAVNEET SINGH, F. THUNG, D. LO, J. LAWALL. *An Empirical Study on the Adequacy of Testing in Open Source Projects*, in "21st Asia-Pacific Software Engineering Conference", Jeju, South Korea, December 2014, https://hal.inria.fr/hal-01096132

[69] T. REPS, T. BALL, M. DAS, J. LARUS. *The Use of Program Profiling for Software Maintenance with Applications to the Year 2000 Problem*, in "ESEC/FSE", 1997, pp. 432–449

[70] L. R. RODRIGUEZ, J. LAWALL. *Increasing Automation in the Backporting of Linux Drivers Using Coccinelle*, in "11th European Dependable Computing Conference - Dependability in Practice", Paris, France, 11th European Dependable Computing Conference - Dependability in Practice, November 2015, https://hal.inria.fr/hal-01213912

[71] C. RUBIO-GONZÁLEZ, H. S. GUNAWI, B. LIBLIT, R. H. ARPACI-DUSSEAU, A. C. ARPACI-DUSSEAU. *Error propagation analysis for file systems*, in "PLDI", Dublin, Ireland, ACM, June 2009, pp. 270–280

[72] L. RYZHYK, P. CHUBB, I. KUZ, E. LE SUEUR, G. HEISER. *Automatic device driver synthesis with Termite*, in "SOSP", 2009, pp. 73–86

[73] L. RYZHYK, A. WALKER, J. KEYS, A. LEGG, A. RAGHUNATH, M. STUMM, M. VIJ. *User-Guided Device Driver Synthesis*, in "OSDI", 2014, pp. 661–676

[74] R. K. SAHA, J. L. LAWALL, S. KHURSHID, D. E. PERRY. *On the Effectiveness of Information Retrieval Based Bug Localization for C Programs*, in "ICSME 2014 - 30th International Conference on Software Maintenance and Evolution", Victoria, Canada, IEEE, September 2014, pp. 161-170 [*DOI :* 10.1109/ICSME.2014.38], https://hal.inria.fr/hal-01086082

[75] R. SAHA, J. L. LAWALL, S. KHURSHID, D. E. PERRY. *On the Effectiveness of Information Retrieval based Bug Localization for C Programs*, in "International Conference on Software Maintenance and Evolution (ICSME)", Victoria, BC, Canada, September 2014

[76] S. SAHA, J.-P. LOZI, G. THOMAS, J. LAWALL, G. MULLER. *Hector: Detecting resource-release omission faults in error-handling code for systems software*, in "DSN 2013 - 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)", Budapest, Hungary, IEEE Computer Society, June 2013, pp. 1-12 [*DOI :* 10.1109/DSN.2013.6575307], https://hal.inria.fr/hal-00918079

[77] D. A. SCHMIDT. *Data Flow Analysis is Model Checking of Abstract Interpretations*, in "POPL", 1998, pp. 38–48

[78] P. SENNA, L. RÉVEILLÈRE, L. JIANG, D. LO, J. LAWALL, G. MULLER. *Understanding the genetic makeup of Linux device drivers*, in "PLOS'13 - 7th Workshop on Programming Languages and Operating Systems", Nemacolin Woodlands Resort, Pennsylvania, United States, ACM, November 2013 [*DOI :* 10.1145/2525528.2525536], https://hal.inria.fr/hal-00927070

[79] P. SENNA TSCHUDIN, J. LAWALL, G. MULLER. *3L: Learning Linux Logging*, in "BElgian-NEtherlands software eVOLution seminar (BENEVOL 2015)", Lille, France, December 2015, https://hal.inria.fr/hal-01239980

[80] M. SHAPIRO. *Purpose-built languages*, in "Commun. ACM", 2009, vol. 52, n^{o} 4, pp. 36–41

[81] R. TARTLER, D. LOHMANN, J. SINCERO, W. SCHRÖDER-PREIKSCHAT. *Feature consistency in compile-time-configurable system software: facing the Linux 10,000 feature problem*, in "EuroSys", 2011, pp. 47–60

[82] F. THUNG, D. X. B. LE, D. LO, J. LAWALL. *Recommending Code Changes for Automatic Backporting of Linux Device Drivers*, in "32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)", Raleigh, North Carolina, United States, IEEE, October 2016, https://hal.inria.fr/hal-01355859

[83] F. THUNG, D. LO, J. L. LAWALL. *Automated library recommendation*, in "WCRE 2013 - 20th Working Conference on Reverse Engineering", Koblenz, Germany, R. LÄMMEL, R. OLIVETO, R. ROBBES (editors), IEEE, October 2013, pp. 182-191 [*DOI :* 10.1109/WCRE.2013.6671293], https://hal.inria.fr/hal-00918076

[84] F. THUNG, S. WANG, D. LO, J. LAWALL. *Automatic recommendation of API methods from feature requests*, in "ASE 2013 - 28th IEEE/ACM International Conference on Automated Software Engineering", Palo Alto, California, United States, E. DENNEY, T. BULTAN, A. ZELLER (editors), IEEE, November 2013, https://hal.inria.fr/hal-00918828

[85] Y. TIAN, D. LO, J. LAWALL. *Automated construction of a software-specific word similarity database*, in "2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE", Antwerp, Belgium, IEEE, February 2014, pp. 44-53, https://hal.inria.fr/hal-01086077

[86] Y. TIAN, D. LO, J. LAWALL. *SEWordSim: software-specific word similarity database*, ACM, May 2014, pp. 568-571, ICSE Companion 2014 - Companion Proceedings of the 36th International Conference on Software Engineering, Poster [*DOI :* 10.1145/2591062.2591071], https://hal.inria.fr/hal-01086079

[87] W. WANG, M. GODFREY. *A Study of Cloning in the Linux SCSI Drivers*, in "Source Code Analysis and Manipulation (SCAM)", IEEE, 2011

[88] S. WANG, D. LO, J. LAWALL. *Compositional Vector Space Models for Improved Bug Localization*, in "30th International Conference on Software Maintenance and Evolution", Victoria, Canada, IEEE, September 2014, pp. 171-180, https://hal.inria.fr/hal-01086084

[89] T. WILLIAMS, P. DAGAND, D. RÉMY. *Ornaments in practice*, in "Proceedings of the 10th ACM SIGPLAN workshop on Generic programming, WGP 2014, Gothenburg, Sweden, August 31, 2014", 2014, pp. 15–24, http://dx.doi.org/10.1145/2633628.2633631

[90] J. YANG, C. HAWBLITZEL. *Safe to the Last Instruction: Automated Verification of a Type-safe Operating System*, in "PLDI", 2010, pp. 99–110