IN PARTNERSHIP WITH:
CNRS

Sorbonne Université (UPMC)

Activity Report 2018

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

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
A2.1.6. - Concurrent programming
A2.1.10. - Domain-specific languages
A2.1.11. - Proof languages
A2.2.1. - Static analysis
A2.2.5. - Run-time systems
A2.3.1. - Embedded systems
A2.3.3. - Real-time systems
A2.4. - Formal method for verification, reliability, certification
A2.4.3. - Proofs
A2.5. - Software engineering
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. Team, Visitors, External Collaborators

Research Scientists

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

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]
Darius Mercadier [Univ Pierre et Marie Curie]
Lucas Serrano [Univ Pierre et Marie Curie]

Technical staff

Antoine Blin [Inria, until Oct 2018, granted by ORANGE SA]

Administrative Assistants

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

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 [37], [42], [80]. 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 [28], [30], [32] or dynamic analysis [51], [53], [59]. 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* [30]. 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* [41] 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 [67]. More recently, more general forms of symbolic execution [28] 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 [25], [26].

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. [35] 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 [29], 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 [17]. 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) [47], 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 [45].

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] [54].

3.1.2.1. Traditional approach.

Using little languages to aid in software development is a tried-and-trusted technique [70] 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, [18] 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 [36] and its descendant Pharo [22] – or multi-paradigm languages – such as the Scala programming language [56] – 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 [19].

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 [27], [23] 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, [24] 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 [20], [21], [57], [60], [64], [66], [68], [52], [46], 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 [73], 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. 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 [23], [27], [55]. 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 [42], 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. [34], numerous research tools have been applied to the Linux kernel, typically for finding bugs [32], [50], [61], [72] or for computing software metrics [39], [78]. In our work, we have studied generic C bugs in Linux code [9], bugs in function protocol usage [43], [44], issues related to the processing of bug reports [65] and crash dumps [38], and the problem of backporting [60], [73], 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, [26] a template-based approach for porting device drivers from one OS to another. Ryzhyk *et al.* propose Termite, [62], [63] 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 [40] 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 [17], Coverity [34], CP-Miner, [49] PR-Miner [50], 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

The Whisper team published three papers at USENIX ATC, one of the major conferences of our domain:

- Coccinelle: 10 Years of Automated Evolution in the Linux Kernel. J. Lawall and G.Muller. [14]
- DSAC: Effective Static Analysis of Sleep-in-Atomic-Context Bugs in Kernel Modules. J.-J. Bai, Y.-P. Wang, J. Lawall, S.-M. Hu. [12]
- The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS. J. Bouron, S. Chevalley, B. Lepers, W. Zwaenepoel, R. Gouicem, J. Lawall, G. Muller, J. Sopena. [13]

Gilles Muller was co-PC chair of DSN 2018, the premier venue for dependable systems.

Julia Lawall was co-PC chair of the ASE 2018 Tool Demo track, in preparation for being the co-PC chair of the main ASE research paper track in 2019.

5.1.1. Awards

The original work on Coccinelle "Documenting and automating collateral evolutions in Linux device drivers" [8] received an ACM EuroSys Test-of-Time award, recognizing it as the paper from EuroSys 2008 that is having the most lasting and current impact (http://eurosys2018.org/awards/).

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/

6.3. Usuba

KEYWORDS: Cryptography - Optimizing compiler - Synchorous language
FUNCTIONAL DESCRIPTION: Usuba is a programming language for specifying block ciphers as well as a bitslicing compiler, for producing high-throughput and secure code.

- Contact: Pierre-Evariste Dagand
- Publication: Usuba, Optimizing & Trustworthy Bitslicing Compiler
- URL: https://github.com/DadaIsCrazy/usuba/

7. New Results

7.1. Software engineering for infrastructure software

The most visible tool developed in the Whisper team is Coccinelle, which this year marked the 10th anniversary of its release in open source. The paper "Coccinelle: 10 Years of Automated Evolution in the Linux Kernel," published at USENIX ATC'18 [14], traced the history of Coccinelle, its underlying design decisions and impact. The Coccinelle C-code matching and transformation tool was first released in 2008 to facilitate specification and automation in the evolution of Linux kernel code. The novel contribution of Coccinelle was to allow software developers to write code manipulation rules in terms of the code structure itself, via a generalization of the patch syntax. Over the years, Coccinelle has been extensively used in Linux kernel development, resulting in over 6000 commits to the Linux kernel, and has found its place as part of the Linux kernel development process. The USENIX ATC paper studies the impact of Coccinelle on Linux kernel development and the features of Coccinelle that have made it possible. It provides guidance on how other research-based tools can achieve practical impact in the open-source development community. This work was also presented to Linux kernel developers at Kernel Recipes and Open Source Summit Europe, and at the 8th Inria/Technicolor Workshop On Systems.

In a modern OS, kernel modules often use spinlocks and interrupt handlers to monopolize a CPU core to execute concurrent code in atomic context. In this situation, if the kernel module performs an operation that can sleep at runtime, a system hang may occur. We refer to this kind of concurrency bug as a sleep-in-atomic-context (SAC) bug. In practice, SAC bugs have received insufficient attention and are hard to find, as they do not always cause problems in real executions. In a paper published at USENIX ATC'18 [12], we propose a practical static approach named DSAC, to effectively detect SAC bugs and automatically recommend patches to help fix them. DSAC uses four key techniques: (1) a hybrid of flow-sensitive and -insensitive analysis to perform accurate and efficient code analysis; (2) a heuristics-based method to accurately extract kernel interfaces that can sleep at runtime; (3) a path-check method to effectively filter out repeated reports and false bugs; (4) a pattern-based method to automatically generate recommended patches to help fix the bugs. We evaluate DSAC on kernel modules (drivers, file systems, and network modules) of the Linux kernel, and on the FreeBSD and NetBSD kernels, and in total find 401 new real bugs. 272 of these bugs have been confirmed by the relevant kernel maintainers, and 43 patches generated by DSAC have been applied by kernel maintainers.

7.2. Trustworthy domain-specific compilers

To achieve safety and composability, we believe that an holistic approach is called for, involving not only the design of a domain-specific *syntax* but also of a domain-specific *semantics*. Concretely, we are exploring the design of *certified domain-specific compilers* that integrate, from the ground up, a denotational and domain-specific semantics as part of the design of a domain-specific language. This vision is illustrated by our work on the safe compilation of Coq programs into secure OCaml code [10]. It combines ideas from gradual typing – through which types are compiled into run-time assertions – and the theory of ornaments [31] – through which Coq datatypes can be related to OCaml datatypes. Within this formal framework, we enable a secure interaction, termed *dependent interoperability*, between correct-by-construction software and untrusted programs, be it system calls or legacy libraries. To do so, we trade static guarantees for runtime checks, thus allowing OCaml values to be safely coerced to dependently-typed Coq values and, conversely, to expose dependently-typed Coq programs defensively as OCaml programs. Our framework is developed in Coq: it is constructive and verified in the strictest sense of the terms. It thus becomes possible to internalize and hand-tune the extraction of dependently-typed programs to interoperable OCaml programs within Coq itself. This work is the result of a collaboration with Eric Tanter, from the University of Chile, and Nicolas Tabareau, from the Gallinette Inria project-team.

7.3. High-performance domain-specific compilers

As part of Darius Mercadier's PhD project, we are developing a synchronous dataflow language targeting high-performance (and, eventually, verified) implementations of bitsliced algorithms, with application to cryptographical algorithms [33]. Using our Usuba language, cryptographers can specify a block cipher at a very high level as a set of dataflow equations. From such a description, our `usubac` compiler is able to generate efficient, vectorized code exploiting the SIMD instruction sets of the underlying architecture. We have demonstrated that our generated code performs on par with hand-tuned assembly programs while, at the same time, being able to target multiple CPU architectures as well as multiple generations of SIMD instruction sets on each architecture. This project illustrates perfectly our methodology: the design of Usuba is driven by semantic considerations (bitslicing is only meaningful for bit parallel operations) that are then structured using types and subsequently reifed into syntactic artefacts. Our preliminary results [15], published in an international workshop, are encouraging.

7.4. Multicore schedulers

As a side-effect of our work on verification of schedulers [48], we have contributed to an analysis of the impact on application performance of the design and implementation choices made in two widely used open-source schedulers: ULE, the default FreeBSD scheduler, and CFS, the default Linux scheduler. In a paper published at USENIX ATC'18 [13], we compare ULE and CFS in otherwise identical circumstances. This work involves porting ULE to Linux, and using it to schedule all threads that are normally scheduled by CFS. We compare

the performance of a large suite of applications on the modified kernel running ULE and on the standard Linux kernel running CFS. The observed performance differences are solely the result of scheduling decisions, and do not reflect differences in other subsystems between FreeBSD and Linux. We found that there is no overall winner. On many workloads the two schedulers perform similarly, but for some workloads there are significant and even surprising differences. ULE may cause starvation, even when executing a single application with identical threads, but this starvation may actually lead to better application performance for some workloads. The more complex load balancing mechanism of CFS reacts more quickly to workload changes, but ULE achieves better load balance in the long run.

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. The PhD of Cédric Courtaud is supported by a CIFRE fellowship as part of this contract.

8.2. Bilateral Grants with Industry

- Oracle, 2018-2019, 100 000 dollars. Operating system schedulers are often a performance bottleneck on multicore architectures because in order to scale, schedulers cannot make optimal decisions and instead have to rely on heuristics. Detecting that performance degradation comes from the scheduler level is extremely difficult because the issue has not been recognized until recently, and with traditional profilers, both the application and the scheduler affect the monitored metrics in the same way.

  The first objective of this project is to produce a profiler that makes it possible to find out whether a bottleneck during application runtime is caused by the application itself, by suboptimal OS scheduler behavior, or by a combination of the two. It will require understanding, analyzing and classifying performance bottlenecks that are caused by schedulers, and devising ways to detect them and to provide enough information for the user to understand the root cause of the issue. Following this, the second objective of this project is to use the profiler to better understand which kinds of workloads suffer from poor scheduling, and to propose new algorithms, heuristics and/or a new scheduler design that will improve the situation. Finally, the third contribution will be a methodology that makes it possible to track scheduling bottlenecks in a specific workload using the profiler, to understand them, and to fix them either at the application or at the scheduler level. We believe that the combination of these three contributions will make it possible to fully harness the power of multicore architectures for any workload.

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 and trustworthy domain-specific compilers.

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.

VeriAmos - awarded in 2018, duration 2018 - 2021

Members: Inria (Antique, Whisper), UGA (Erods)

Coordinator: Xavier Rival

Whisper members: Julia Lawall, Gilles Muller

Funding: ANR, 121,739 euros.

Objectives:

General-purpose Operating Systems, such as Linux, are increasingly used to support high-level functionalities in the safety-critical embedded systems industry with usage in automotive, medical and cyber-physical systems. However, it is well known that general purpose OSes suffer from bugs. In the embedded systems context, bugs may have critical consequences, even affecting human life. Recently, some major advances have been done in verifying OS kernels, mostly employing interactive theorem-proving techniques. These works rely on the formalization of the programming language semantics, and of the implementation of a software component, but require significant human intervention to supply the main proof arguments. The VeriAmos project will attack this problem by building on recent advances in the design of domain-specific languages and static analyzers for systems code. We will investigate whether the restricted expressiveness and the higher level of abstraction provided by the use of a DSL will make it possible to design static analyzers that can statically and fully automatically verify important classes of semantic properties on OS code, while retaining adequate performance of the OS service. As a specific use-case, the project will target I/O scheduling components.

9.3. International Initiatives

9.3.1. Inria International Labs

- EPFL-Inria Lab Our work on scheduling [13] and on the Ipanema DSL [48] is done as part of the EPFL-Inria Lab. Our direct partners, Willy Zwaenepoel and Baptiste Lepers, have moved to the University of Sydney in September 2018. Therefore we have migrated our cooperation.

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 [58], [69], [71], [74], [75], [76], [79], [77], at conferences including ASE and ICSME. The ITrans ANR is a joint project with them.
- We collaborate with David Lo and James Hoang of Singapore Management University and with Sasha Levin of Microsoft on the use of machine learning to identify stable-relevant patches in the Linux kernel. Preliminary results from this collaboration have been presented with Sasha Levin at the Open Source Summit North America, the Open Source Summit Europe, and the Linux Plumbers Conference kernel summit track.
- Our previous collaboration with EPFL has been transfered to the University of Sydney due to the moves of Willy Zwaenepoel and Baptiste Lepers.
- We collaborate with Christoph Reichenbach of the University of Lund and Krishna Narasimhan of Itemis (Germany) on program transformation and the design of tools for code clone management [11].
- We collaborate with Jia-Ju Bai of Tsinghua University on bug finding in Linux kernel code, particularly focusing on issues requiring interprocedural analysis [12].
- As part of the LIP6 Invited Professor program, we have initiated a collaboration between Karine Heydeman (ALSOC team – LIP6, France) and Patrick Schaumont (Virginia Tech, US) on the development of fault-resistant and side-channel attack resistant compilation techniques.

9.4. International Research Visitors

9.4.1. Visits of International Scientists

- Patrick Schaumont of Virginia Tech visited LIP6 in July and November 2018, as part of the LIP6 Invited Professor program.
- David Lo and Lingxiao Jiang of Singapore Management University visited the Whisper team for two weeks in October 2018 as part of the ANR ITrans project.
- Michele Martone of the Leibniz Supercomputing Centre in Munich Germany made two visits of one week each to the Whisper team in August and December to work on applying Coccinelle to high performance computing code.

9.4.1.1. Internships

- Jonathan Carroll of Oberlin College spent January 2018 working on using machine learning to identify stable-relevant patches for the Linux kernel.
- David Bergvelt of the University of Illinois at Urbana-Champaign spent May-August 2018 working on applying Verifiable C, developed at Princeton, to verification of process schedulers.

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Selection

10.1.1.1. Chair of Conference Program Committees

- Gilles Muller: DSN 2018
- Julia Lawall: ASE 2018 Tool Demo track.

10.1.1.2. Member of the Conference Program Committees

- Gilles Muller: OSDI 2018, EuroSys 2018
- Julia Lawall: EuroSys 2018, ICSE-NIER 2018, ASPLOS 2018 ERC, PEPM 2018, SCAM 2018, APSys 2018, USENIX ATC 2018, CARI 2018

10.1.2. Journal

10.1.2.1. Member of the Editorial Boards

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

10.1.2.2. Reviewer - Reviewing Activities

- Julia Lawall: Transactions on Software Engineering, Software: Evolution and Process, IEEE Transactions on Reliability, ACM Transactions on Embedded Computing Systems

10.1.3. Invited Talks

- Gilles Muller:
  - "Provably Work Conserving Multicore Schedulers", University of Bordeaux, June 13, 2018.
  - "Safe multicore scheduling in a Linux cluster environment", 3rd GDR RSD and ASF Winter School on Distributed Systems and Networks, Sept Laux, March 20, 2018.
- Julia Lawall:
  - "Coccinelle: 10 Years of Automated Evolution in the Linux Kernel", 8th Inria/Technicolor Workshop On Systems, Rennes, December 11, 2018.
  - "Software evolution and bug finding using Coccinelle", Lightweight analysis and verification techniques, Verimag, Grenoble, December 11, 2018.
  - "Coccinelle: 10 Years of Automated Evolution in the Linux Kernel", Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS), Toulouse, July 3, 2018.
  - "Coccinelle: Practical Program Transformation for the Linux Kernel", EJCP 2018 : École Jeunes Chercheurs et Jeunes Chercheuses en Programmation 2018, June 25, 2018.
  - "Introduction to Coccinelle and its usage in the Linux Kernel", Conférence MiNET, Telecom SudParis, May 24, 2018.
- Pierre-Évariste Dagand gave a seminar at the Collège de France entitled "Types dépendants : tout un programme" (November 28, 2018), as part of Xavier Leroy's chair "Sciences du logiciel".
- Lucas Serrano: "Inference of Semantic Patches from Code Examples", The Seventh International Workshop on Software Mining, with ASE, September 3, 2018.
- Cedric Courtaud "Toward an Efficient Data Plane for Memory Systems Interference Regulation in COTS Multi-core Systems", The NExt Step TOwards multi-core Real-time systems workshop, ULB, May 18, 2018.

10.1.4. Scientific Expertise

- Julia Lawall was part of the midterm review panel of the NSF Expedition in Computing project DeepSpec.

10.1.5. Research Administration

- Julia Lawall: IFIP TC secretary (2012 - present). Elected member of IFIP WG 2.11 (Program Generation).

  Member of a hiring committee for a Maître de conférences position at Université Paris Diderot

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

- Gilles Muller: 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.
- 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
- Master: Pierre-Évariste Dagand, Specification and Validation of Programs, M2, UPMC, France
- Licence: Pierre-Évariste Dagand, INF311: Introduction to Programming, L1, École Polytechnique, France
- Master: Pierre-Évariste Dagand, INF559: Computer Architecture and Operating Systems, M1, École Polytechnique, France

10.2.2. Supervision

- PhD : Mariem Saeid, soutenue le 25/9/2018, Jens Gustedt (Camus), Gilles Muller.
- PhD in progress : Cédric Courtaud, CIFRE Thalès, 2016-2019, Gilles Muller, Julien Sopéna (Delys).
- PhD in progress : Redha Gouicem, 2016-2019, Gilles Muller, Julien Sopéna (Delys).
- 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

- Julia Lawall: PhD juries of Ferdian Thung, SMU (reporter), Thibaut Girka, Université Paris Diderot (president), Thomas Durieux, Lille (examiner).

10.3. Popularization

- Julia Lawall: Coordinator of the Outreachy internship program for the Linux kernel, until March 2018. Outreachy provides remote 3-month internships twice a year for women and other underrepresented minorities on open source projects. Julia Lawall also mentored Aishwarya Pant as part of this program.
- Julia Lawall, "Building Stable Trees with Machine Learning", Open Source Summit North America, August 2018, with Sasha Levin. Open Source Summit Europe, October 2018, with Sasha Levin. Linux Plumbers Conference, kernel summit track, November 2018, with Sasha Levin.
- Julia Lawall, "Coccinelle: 10 Years of Automated Evolution and Bug Finding in the Linux Kernel", Open Source Summit Europe, October 2018.
- Julia Lawall, "Panel Discussion: Outreachy Kernel Internship Report" (moderator), Open Source Summit Europe, October 2018.
- Julia Lawall, "Panel Discussion: An Exploration of Insights & Issues Related to Mentoring Programs" (participant), Open Source Summit Europe, October 2018.
- Julia Lawall, "Interprocedural Static Analysis Strategies for the Linux Kernel: Detecting SAC Bugs as an Example (Work in Progress)", Linux Kernel Real Time Summit, October 2018.
- Julia Lawall, "Kernel Panel" (participant), Linux Plumbers Conference, November 2018.

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° 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° 2, pp. 4:1–4:40

Publications of the year

Articles in International Peer-Reviewed Journals

[10] P.-E. DAGAND, N. TABAREAU, É. TANTER. *Foundations of Dependent Interoperability*, in "Journal of Functional Programming", March 2018, vol. 28 [*DOI :* 10.1017/S0956796818000011], https://hal.inria.fr/hal-01629909

[11] K. NARASIMHAN, C. REICHENBACH, J. LAWALL. *Cleaning up Copy-Paste Clones with Interactive Merging*, in "Journal of Automated Software Engineering", August 2018, vol. 25, n° 3, pp. 627-673 [*DOI :* 10.1007/s10515-018-0238-5], https://hal.inria.fr/hal-01853896

International Conferences with Proceedings

[12] J.-J. BAI, Y.-P. WANG, J. LAWALL, S.-M. HU. *DSAC: Effective Static Analysis of Sleep-in-Atomic-Context Bugs in Kernel Modules*, in "2018 USENIX Annual Technical Conference", Boston, MA, United States, July 2018, https://hal.inria.fr/hal-01853268

[13] J. BOURON, S. CHEVALLEY, B. LEPERS, W. ZWAENEPOEL, R. GOUICEM, J. LAWALL, G. MULLER, J. SOPENA. *The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS*, in "2018 USENIX Annual Technical Conference", Boston, MA, United States, July 2018, https://hal.inria.fr/hal-01853267

[14] J. LAWALL, G. MULLER. *Coccinelle: 10 Years of Automated Evolution in the Linux Kernel*, in "2018 USENIX Annual Technical Conference", Boston, MA, United States, July 2018, https://hal.inria.fr/hal-01853271

[15] D. MERCADIER, P.-É. DAGAND, L. LACASSAGNE, G. MULLER. *Usuba, Optimizing & Trustworthy Bitslicing Compiler*, in "WPMVP'18 - Workshop on Programming Models for SIMD/Vector Processing", Vienna, Austria, ACM Press, February 2018 [*DOI :* 10.1145/3178433.3178437], https://hal.archives-ouvertes.fr/hal-01657259

Other Publications

[16] M. MARTONE, L. IAPICHINO, N. HAMMER, J. LAWALL. *Automating Data Layout Conversion in a Large Cosmological Simulation Code*, September 2018, CoSaS 2018 - International Symposium on Computational Science at Scale, Poster, https://hal.inria.fr/hal-01890314

References in notes

[17] 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

[18] 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

[19] 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

[20] 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° 2, pp. 177-196, https://hal.archives-ouvertes.fr/hal-00844536

[21] 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

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

[23] 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

[24] 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

[25] 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

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

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

[28] 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

[29] 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

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

[31] P.-É. DAGAND. *Reusability and Dependent Types*, University of Strathclyde, 2013

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

[33] 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

[34] 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

[35] 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

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

[37] 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

[38] 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

[39] 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

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

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

[42] 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

[43] 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

[44] 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

[45] 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

[46] 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

[47] 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

[48] 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

[49] 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

[50] 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

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

[52] 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

[53] 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

[54] 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

[55] 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

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

[57] 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

[58] 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

[59] 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

[60] 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

[61] 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

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

[63] 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

[64] 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

[65] 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

[66] 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

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

[68] 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

[69] P. SENNA TSCHUDIN, 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

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

[71] 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

[72] 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

[73] 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

[74] 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

[75] 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

[76] 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

[77] 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

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

[79] 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

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