Reprinted from *Operating Systems Review* 43(4):31–34, December 2009. http://doi.acm.org/10.1145/1713254.1713262

PLOS 2009: Fifth Workshop on Programming Languages and Operating Systems

Workshop Overview*

Eric Eide
University of Utah
eeide@cs.utah.edu

Gilles Muller
INRIA REGAL
Gilles.Muller@lip6.fr

Olaf Spinczyk
TU Dortmund
olaf.spinczyk@udo.edu

ABSTRACT

This report summarizes the Fifth Workshop on Programming Languages and Operating Systems (PLOS 2009), which was held in conjunction with the SOSP 2009 conference. This report presents the motivation for holding the workshop and summarizes the workshop contributions.

1. INTRODUCTION

The goal of the PLOS workshop series is to bring together researchers and developers from the programming languages (PL) and the operating systems (OS) domains to discuss recent work at the intersection of these fields. PLOS is a platform for discussing visions, challenges, experiences, problems, and solutions arising from the application of advanced programming and software engineering concepts to operating systems construction, and vice versa.

In 2009, the PLOS workshop was held in conjunction with SOSP. This edition of the workshop was very successful—probably the most successful PLOS workshop ever! Thirty-nine participants attended high-quality paper presentations, saw inspiring demonstrations, and participated in a highly interactive workgroup session.

2. MOTIVATION

Historically, OS development and programming language development went hand-in-hand. Cross-fertilization was the norm. Challenges in one area were often approached using ideas or techniques developed in the other, and advances in one enabled new capabilities in both. Today, although the systems community at large retains an iron grip on C, novel approaches to OS construction based on new programming language ideas continue to be an active and important area of research. The systems field continues to provide a wealth of challenge problems and new results that should spark advances in programming languages, software designs, and idioms.

The connection between OS development and programming languages is both significant and current. This is demonstrated by operating systems such as seL4 [5] and Singularity [3], embedded OS frameworks such as TinyOS [7], OS extension frameworks such as SafeDrive [11], and the "programming language techniques" session at OSDI 2008. The PLOS workshop series is intended to be a venue for new and emerging research that follows in the footsteps of these examples—new ideas that further explore the synthesis of programming language and operating system concepts.

3. PREPARATION

Eric Eide, Andreas Gal, Gilles Muller, and Olaf Spinczyk proposed to hold the fifth PLOS workshop in conjunction with SOSP 2009. The proposal was accepted, and the organizers quickly published a call for workshop papers. Suggested topics for submissions to PLOS 2009 were:

- critical evaluations of new programming language ideas in support of OS construction;
- domain-specific languages for operating systems;
- type-safe languages for operating systems;
- object-oriented and component-based operating systems;
- language-based approaches to crosscutting system concerns, such as security and run-time performance;
- language support for system verification;
- language support for OS testing and debugging;
- static/dynamic configuration of operating systems;
- static/dynamic specialization within operating systems; and
- the use of OS abstractions and techniques in language runtimes.

Gilles Muller was the PLOS 2009 program chair. The other members of the program committee were:

- Eric Eide, University of Utah
- Manuel A. Fähndrich, Microsoft Research
- Andreas Gal, Mozilla Corporation
- Robert Grimm, New York University
- Gernot Heiser, University of New South Wales
- Maurice Herlihy, Brown University
- Urs Hölzle, Google
- Bernd Mathiske, Adobe
- Laurent Réveillère, ENSEIRB/LABRI

The program committee received fourteen paper submissions. Following a rigorous review process, the committee selected seven for presentation at the workshop.

4. PROGRAM

The PLOS 2009 program is shown in Figure 1. The day began with an invited keynote address by David Gay, who described the Ivy research project. Following the keynote, the first workshop session was devoted to the papers presenting programming-language approaches to the domains of kernels and distributed systems. The second session started in the afternoon and addressed the topic of domain-specific languages (DSLs) for systems programming. In the third workshop session, several authors demonstrated the systems described in their papers. This allowed the PLOS participants to interact widely and ask detailed questions. During the last session, the workshop participants organized working groups to discuss two topics in depth: (1) formal verification and DSLs, and (2) programming-language annotations.

The paper *"Filet-o-Fish: Practical and Dependable Domain-Specific Languages for OS Development"* by Pierre-Evariste Dagand, Andrew Baumann, and Timothy Roscoe was chosen as the

*http://www.ploswo rkshop.org/2009/

- **Welcome and Keynote**
  - Keynote Address: *Ivy: Modernizing C*
    David Gay (Intel Research Berkeley)
- **Session 1: Papers about Kernels and Distributed Systems**
  - *Checking Process-Oriented Operating System Behaviour using CSP and Refinement*
    Frederick R. M. Barnes and Carl G. Ritson (University of Kent)
  - *A Microkernel API for Fine-Grained Decomposition*
    Sebastian Reichelt, Jan Stoess, and Frank Bellosa (University of Karlsruhe)
  - *Code-Partitioning Gossip*
    Lonnie Princehouse and Ken Birman (Cornell University)
  - *CatchAndRetry: Extending Exceptions to Handle Distributed System Failures and Recovery*
    Emre Kıcıman, Benjamin Livshits, and Madanlal Musuvathi (Microsoft Research)
- **Session 2: Papers about Domain-Specific Languages**
  - *Filet-o-Fish: Practical and Dependable Domain-Specific Languages for OS Development* **— (Best Paper)**
    Pierre-Evariste Dagand (ENS Cachan-Bretagne), Andrew Baumann, and Timothy Roscoe (ETH Zurich)
  - *KStruct: Preserving Consistency Through C Annotations*
    Alexander Schmidt, Martin von Löwis, and Andreas Polze (Hasso Plattner Institute at University of Potsdam)
  - *Distributed Data Flow Language for Multi-Party Protocols*
    Krzysztof Ostrowski, Ken Birman (Cornell University), and Danny Dolev (Hebrew University)
- **Session 3: Demonstrations**
- **Session 4: Working Groups and Wrap Up**

Figure 1: Workshop Program

best paper of PLOS 2009 by the workshop attendees.

4.1 Keynote

David Gay from Intel Research Berkeley summarized his work toward improving the C programming language. His approach is based on adding annotations to C code—both legacy and new C programs—to help prevent the occurrence of bugs. Three approaches and associated annotation systems have been developed as part of the Ivy research project. The first, embodied in the *Deputy* compiler, seeks to prevent the occurrence of type-safety and memory-safety errors. The second, implemented in *HeapSafe*, is for preventing the problems caused by dangling pointers. The third, implemented in the *SharC* tool, is for the safe sharing of complex data structures in concurrent programs. These approaches have been used to improve the safety of the Linux kernel and TinyOS. The Ivy tools are publicly available at `http://ivy.cs.berkeley.edu/`.

4.2 Session 1: Kernels & Distributed Systems

Starting the first session, Frederick Barnes from the University of Kent presented a CSP-based approach to kernel development. In this approach, a kernel is built from a set of concurrent processes that communicate through well-defined channels; in addition, these channels can be reconfigured dynamically. The occam-pi programming language allows systems to be implemented using these abstractions. The advantage of Fred's approach is that communications and process composition can be formally verified at compile time. A example kernel, *RMoX*, was developed following these principles and partially verified.

Sebastian Reichelt from the University of Karlsruhe then described an approach for developing a microkernel using fine-grained, isolated components. The motivation for this work is to offer a programming model close to traditional monolithic systems, and at the same time, to support modularity like that found in microkernels. The main advantage of this solution is to permit the reuse of existing systems code with little reengineering. As a demonstration of such reuse, a prototype OS has been built by reusing Linux drivers and the LwIP networking stack.

In the third talk of the session, Lonnie Princehouse from Cornell University presented a promising approach for simplifying the development of gossip protocols. The idea is to describe an overall protocol as a collection of pairwise node-transaction functions, written in Java. A pairwise function describes how the states of two communicating nodes are (atomically) updated by a gossip exchange. A code slicer transforms these functions into the implementations of the gossip initiator and gossip recipient. Proxies and lower-level networking code are generated automatically.

The last talk of the session was given by Ben Livshits from Microsoft Research. He described a set of language extensions for expressing retry-based recovery strategies. When an operation in a distributed system fails, it is often possible and useful for the system to recover by simply retrying the operation. Ben presented a set of language constructs for describing retry strategies in a concise manner. With these new constructs, a block of code can be retried several times, possibly with a modified environment each time. Retries can also be delayed until some additional condition is satisfied. Using several realistic examples from Facebook, Ben described uses of his retry mechanisms to cope with failures.

4.3 Session 2: Domain-Specific Languages

The afternoon paper session was devoted to domain-specific languages and their uses in implementing operating systems.

Pierre-Evariste Dagand from ENS Cachan-Bretagne described *Filet-o-Fish*, an approach to simplifying the implementation of DSLs. Filet-o-Fish supports both the specification of the semantics and the generation of C code in the back end of the compiler. Filet-o-Fish is a first step toward a verified DSL compiler. It has been developed in the context of the Barrelfish operating system [1].

Alexander Schmidt then presented *KStruct*, an annotation-based language for the consistent monitoring of data structures. The language is based on notations already introduced in Microsoft Windows for declaring lock behavior. KStruct annotations are used at compile time to generate monitoring code. At run time, the monitors access data while masking unwanted intermediate state.

The last talk was given by Krzysztof Ostrowski from Cornell, who presented a novel object-oriented model for defining the semantics of distributed multi-party protocols. Multi-party protocols—including ones for leader election, distributed locks, and reliable multicast—are commonly used in peer-to-peer systems. Krzysztof introduced the concept of a *distributed flow*, which captures high-level protocol semantics concisely. The notion of a distributed flow can reduce coding burden, support reasoning about the protocol behavior, and provide a high degree of architectural flexibility.

4.4 Demonstration Session

Following the paper presentations, the PLOS attendees turned their attention to demonstrations. Four of the systems described in the paper presentations were demonstrated during the workshop. Fred Barnes presented the RMoX operating system running on a laptop and on two embedded platforms. At the same time, Sebastian Reichelt demonstrated tools for working with the microkernel architecture and components that he had described during his talk. Pierre-Evariste Dagand demonstrated Filet-o-Fish. Finally, Krzysztof Ostrowski presented his distributed data flow language.

4.5 Working Group: DSLs and Verification

Following the demonstrations, the workshop participants organized themselves into two working groups to focus in-depth on topics that had emerged during the workshop. The first working group discussed the relationship between domain-specific languages and the formal verification of operating systems. The paragraphs below summarize some of the ideas that came from this working group.

Translation validation for dependable DSLs. For Filet-o-Fish to be truly dependable, it would need to be formalized in a theorem prover. The Agda functional language [10] is being seriously considered for this purpose. Using Agda, the functional semantics of Filet-o-Fish would not consist of Haskell functions, which cannot be formally reasoned about, but mathematical objects, which could be formally manipulated. It has been commonly agreed that this objective could be achieved with relative ease.

After seL4, what's next? Although the verification of seL4 is finished [5], the seL4 developers have mentioned some interesting research issues and directions they are considering. For example, they are using DSLs to automate repetitive tasks and derive a correctness proof of the generated code. Moreover, an important issue remains: determining the trusted computing base (TCB) and ensuring its correctness. Beyond the TCB, isolating or restricting the interaction between components executing on top of the kernel is also a challenging problem.

Measuring operating-system dependability. Although the verification of systems code is flourishing, the community lacks a set of metrics to compare these works. The Common Criteria [8] were meant to solve this problem. They define seven levels of correctness, each consisting of a set of properties. However, in practice, people tend to implement and verify their systems up to some level, and then grab the low-hanging fruits in various higher-graded components. To compare verified systems, therefore, researchers need to compare the verified properties of their systems to those in other systems, including performance, and the various flavors of certain correctness invariants.

Going down to assembly. Because seL4 correctness is ensured only down to the level of C code, one must trust the correctness of the C compiler. The CompCert certified compiler [6] comes to mind as a way to gain trust below the level of C code. However, "simply" compiling seL4 with CompCert would be of relatively small benefit. The ideal situation would be to have a correctness proof mapping the high-level seL4 invariants down to the assembly code. This approach is being explored.

Not much is lost by limiting to C, however: "C is the universal assembly." In practice, seL4 had a requirement to perform within 10% of the speed of L4, which is met—seL4 is sometimes even faster than L4. Thus, performance has not been a problem. In term of effort, the correctness proof from the Haskell model to C was reported to be significantly easier than that from the Isabelle model to Haskell.

Integrating domain-specific logics. Bossa [9], a DSL for writing schedulers, was mentioned during the working group. By using this DSL, a novice is able to write a scheduler for Linux that, provably, will not crash. This guarantee, among others, is ensured by abstractions that are tailored to the problem at hand. This example embodies the importance of the logic of DSLs, beyond their mere syntactic aspect. A DSL is not only useful for decreasing the amount of code that one must write. A good DSL also captures the logic of the problem—that is, the semantics and the invariants that should be maintained.

Dependable DSL parsers. A common issue for a certified compiler is that it relies on the correctness of its parser. The seL4 operating system and Filet-o-Fish are no exceptions. To the knowledge of the working group, it is an open problem to be able to guarantee the correctness of a parser. Translation validation comes to mind, but one might need to worry about the speed of the resulting parser.

4.6 Working Group: Annotations

The second working group discussed the use of source-level annotations to improve systems programming languages. Many emerging tools rely on programmers to insert annotations into their source code as a way of attaching properties to parts of their programs. Typically, a tool defines a unique set of annotations for itself, and depending on the tool's purpose, it may require a programmer to insert few or many annotations into a system's source code. If many different tools are used at once, then many different types of annotation may need to be inserted into a system's source code.

In his PLOS workshop keynote, David Gay colorfully referred to the overuse of annotations as "annotation diarrhea." The second PLOS working group sought to explore the use of annotations for systems programming and seek a cure for this new-found disease.

The first question discussed was, **Why are annotations so widely used in research systems?** An important argument was that annotations can compensate for some of the language deficiencies of C. The annotations supported by Ivy are a good example of this. By reducing the likelihood of bugs and security vulnerabilities in C code, it might be possible to keep the language alive longer. (The group did not discuss if "keeping C alive" is truly a worthy goal.) Annotations can also ease software development and are very powerful if they extend the language's type system, e.g., through user-defined type qualifiers. Another point of view was that annotations are a vehicle to express the programmer's intent explicitly in the source code. This is useful not only as a documentation, but also as an important input to a compiler, aspect weaver, or arbitrary transformation system.

Another question was, **Which annotation mechanisms are out there, and what are their pros and cons?** The working group discussed many different approaches including the standard mechanisms in Java and .NET and the established mechanisms in C, e.g., pragmas, the OpenMP API [2], and GCC attributes. The group also catalogued more sophisticated mechanisms such as the metadata systems of modern dynamic languages like Clojure [4] and Python.

The last question was, **How can the community help get rid of annotation diarrhea?** Having many different and incompatible annotation systems is counterproductive. In the long term, a standard mechanism for the family of C-based languages should be defined. In addition, the amount of necessary annotation should be reduced by having tools incorporate reasonable defaults. It was also proposed to move annotations "out of the file"—instead of putting annotations in the source code, one could rely on the programming environment to keep track of annotations and their attachments to source-level constructs. A modern IDE could certainly handle this. However, this suggestion was controversial, and the practitioners insisted that developers would not accept being tied to any particular IDE.

The discussions of the two working groups converged when the PLOS attendees noted that an annotation mechanism is a kind of lightweight and "sneaky" DSL that is embedded into a general-purpose language.

5. CONCLUSION

With almost forty participants who enjoyed a lively agenda of research presentations, system demonstrations, and working groups, the Fifth Workshop on Programming Language and Operating Systems was a great success. The organizers believe that they achieved their goal, to provide a venue for emerging research at the intersection of OS development and programming language development. They hope to see you at a future edition of PLOS!

6. ACKNOWLEDGMENTS

We thank Chris Matthews and Pierre-Evariste Dagand for taking excellent and detailed notes during the workshop. Their work contributed directly to this report. We also give special thanks to Marc Fiuczynski and Jeanna Matthews for the many hours that they devoted to the organization of the SOSP 2009 workshop program.

7. REFERENCES

[1] A. Baumann et al. The multikernel: A new OS architecture for scalable multicore systems. In *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 29–44, Big Sky, MT, Oct. 2009.

[2] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2007.

[3] M. Fähndrich et al. Language support for fast and reliable message-based communication in Singularity OS. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 177–190, Leuven, Belgium, Apr. 2006.

[4] S. Halloway. *Programming Clojure*. Pragmatic Bookshelf, May 2009.

[5] G. Klein et al. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 207–220, Big Sky, MT, Oct. 2009.

[6] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, July 2009.

[7] P. Levis et al. T2: A second generation OS for embedded sensor networks. Technical Report TKN–05–007, Telecommunication Networks Group, Technische Universität Berlin, Nov. 2005.

[8] Members of the CCRA. The Common Criteria portal, 2009. `http://www.commoncriteriaportal.org/thecc.html`.

[9] G. Muller, J. L. Lawall, and H. Duchesne. A framework for simplifying the development of kernel schedulers: Design and performance evaluation. In *HASE 2005 - High Assurance Systems Engineering Conference*, pages 56–65, Heidelberg, Germany, Oct. 2005.

[10] The Agda Team. The Agda wiki — Adga, 2009. `http://wiki.portal.chalmers.se/agda/agda.php`.

[11] F. Zhou et al. SafeDrive: Safe and recoverable extensions using language-based techniques. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 45–60, Seattle, WA, Nov. 2006.