Microlanguage-Based Specialization

Technical Report for April – June 1996

Abstract

We summarize the progress made during the period April through June 1996 for the DARPA contract "Microlanguage-Based Specialization", part of the Synthetix Project. The main objective of the project is to design and implement microlanguages to support *directed* specialization of operating system kernels, to complement the *inferred* specialization in the Synthetix project. The key idea is to design a microlanguage tailored for each specific area that captures deep application semantics through a simple syntax. We are making significant progress in the construction of the Synthetix specialization toolkit and the design of the first family of microlanguages for distributed multimedia support. We also report on the progress made in the context of the entire Synthetix project.

DARPA Order Number: D007 and D308

Name of contractor: Oregon Graduate Institute of Science & Technology
P.O. Box 91000
Portland, OR 97291-1000

Contract Number: F19528-95-C-0193

Principal Investigator: *Calton Pu*
Phone: (503) 690-1214
FAX: (503) 690-1553
Email: calton@cse.ogi.edu
URL: http://www.cse.ogi.edu/~ calton

Project Title: Microlanguages (Synthetix Project)

Sponsored by
Defense Advanced Research Projects Agency, CSTO/ITO
DARPA Order No. D007 and D308
Issued by ESC/ENS under contract F19528-95-C-0193

Contents

1 Task Objectives

1.1 Summary of Project Objectives

This contract (F19528-95-C-0193) strengthens and complements the Synthetix project grant (DARPA/ONR grant N00014-94-1-0845). Although the contract has just started in October 1995, we have achieved significant progress from the ongoing Synthetix work. We will report the progress and achievements of the project as a whole, distinguishing the results from each contract funding as appropriate.

For the Synthetix grant, the concrete deliverables fall into three categories: a methodology for applying it to existing operating system kernels, tools to support incremental specialization, and kernel code that demonstrates its use in practice. Similarly, for the Microlanguages contract, the concrete deliverables fall into three groups: techniques, microlanguages plus their associated software tools, and experimental systems that use microlanguages.

Synthetix specialization is primarily targeted towards situations where the systems software can *infer* the kind of specialization to apply. The goal of microlanguages is to expand the power and applicability of specialization techniques and toolkit to situations where systems programmers and application programmers may *direct* the specialization process. Some of the basic specialization techniques and tools apply equally to inferred and directed specialization, but many methods and tools need to be refined and redesigned to take into account both kinds of specialization.

1.2 Synthetix Deliverables

We will develop a methodology for the design and implementation of operating systems (and their components) that use incremental specialization. Concretely, this methodology will outline a series of steps that will guide kernel developers both in the development of new kernels from scratch, and in the modification of existing kernel code. The methodology will apply to existing monolithic kernels and micro-kernels.

We will develop a toolkit that includes a C preprocessor and compiler with support for: (a) fine-grain modularity, (b) the specification of invariants and guards, and (c) the generation of templates that can be instantiated post compile time. In addition, the toolkit will also include a run-time kernel supporting dynamic code generation, dynamic linking, and possibly some code optimization such as constant folding and loop unrolling.

Another important part of Synthetix specialization toolkit is the support for fine-grain adaptation using software feedback. The software feedback toolkit will include (1) software implementation of generic filters to be used in the kernels, (2) software tools for filter design, development, and testing, (3) composition tools to combine elementary filters into more sophisticated filters, and (4) guard programs that detect input oscillation beyond the specified filter range.

The specialization toolkit will be demonstrated in production operating system kernels. Our initial experiments were conducted on the HP-UX, the HP commercial version of Unix. Current experiments are being conducted on Linux, a public domain Unix operating system.

1.3 Microlanguage Deliverables

The first group of Microlanguage deliverables includes the expanded research methodology on specialization and microlanguage design strategies. The design of a microlanguage is complemented by microcompilers that generate code and microengines that execute the compiled microprograms. We will develop techniques to integrate the specialized underlying operating system kernel and the specialized microengine, to minimize the overhead of running the microprograms.

The second group of Microlanguage deliverables includes the microlanguages themselves and the software tools that support the microlanguages such as microcompilers and microengines. We have been using a DARPA-community compiler toolkit (SUIF of Stanford) for the development of incremental specialization tools in the Synthetix project. Our main contribution will be the techniques and tools to help microlanguage designers to handle deep application semantics.

The third group of Microlanguage deliverables consists of system components that we use to experiment with microlanguages, evaluate their performance as well as flexibility, and to demonstrate the advantages of microlanguages. For example, the experimental evaluation of first microlanguage for file systems may be conducted on a simplified version of Unix File System.

We plan to have deliverables from each group (with increasing degrees of sophistication, robustness, and variety) for the Base Period, Option 1, and Option 2 of the Microlanguages contract.

2 Technical Problems

Modern operating systems have been growing in size and complexity due to the constant pressure for additional functionality. As the variety of applications widens, and hardware platforms become increasingly powerful, the operating system has been used for different purposes. Consequently, operating system code has been stretched far beyond its original intent. For example, file systems optimized for block size access typically do not support byte-by-byte access efficiently. Although libraries such as `stdio` keep work outside the kernel, the result is the growing size and complexity of both the kernel code and the systems software related to the kernel.

To aggravate the problem, kernel operations are encapsulated within blackbox boundaries, so users cannot customize the kernel very easily. Specialization of the operating system

kernel has been developing as a promising technique in a number of projects such as Synthesis [17, 14], x-kernel [15], Spin [1], and Synthetix [6, 16]. However, specialization has encountered two limitations in its wide application. First, it has been applied successfully in specific domains, but each domain seems different enough to require a large new effort. Second, efforts to make customization easy, such as Spin, have yet to address the concerns of quality control, interference with other kernel modules, maintainability, and system evolution.

We propose to apply discipline to specialization through *microlanguages*, meta-interface languages that characterize how an interface is used, and tailored for each application. Each microlanguage specifies all the specialized execution paths in an appropriate systems software component, e.g., file system, network protocol, and a client/server interface for an application area. On the client's side, a sufficiently rich microlanguage provides a high level declarative way to express customization, rather than by writing code in (third-generation) system implementation language that then has to be "sanitized" and inserted into the kernel. On the systems side, kernel designers determine the scope of specialization during the design and implementation of microlanguages.

The microlanguage approach can be explained intuitively by an analogy with the highway system. Before the highway system was invented, driving long distance was an adventure that connected from one local road to another local road. Trucking was a business that contained a large amount of uncertainty, since road conditions were unpredictably variable (e.g., due to local maintenance constraints) and the long distance traffic interfered with many local communities. Execution paths within operating systems have similar patterns to such road traffic patterns.

In our vision, instead of directly providing the user with a third-generation programming language without restrictions to customize the kernel, we introduce specialized and limited microlanguages that allow users to program specific functions in the kernel (and higher levels) in a safe, structured and disciplined way. The same way most drivers rather use existing roads (if available) instead of building their own, we think that most programmers prefer to specify what they want instead of writing new code that must be debugged and maintained. Microlanguages are a structured and disciplined way of "programming" a highly-parameterized (sub-)system. As the number of parameters and knobs increase in the system, x-kernel style microprotocol approaches become more difficult, since static parameters alone may not conveniently describe all the situations.

3 General Methodology

Our approach can be divided into four stages. The first stage is the design of a microlanguage for a chosen system software component (or layer). The design goal of a microlanguage is to allow a minimal set of primitives, parameters, and constraints to be expressed in a controlled

and simple way. Using our highway analogy, the designer of a microlanguage is laying out a highway system for that area. It is easy to see that we want to reduce the investment made in building the highway system, while covering the area well.

The design of microlanguages depends on the application semantics of the chosen area, and also on the current usage of the application. The same way a good highway system serves better an area more densely populated, we will support finer granularity control for execution paths more frequently traversed. Since the usage patterns may change over time, it is important to make microlanguages extensible, since it will be maintained continuously, just like highway systems. By focusing on parameters, declarations, and constraints, we want to maximize the extensibility of microlanguages, since these constructs tend to have high orthogonality.

The second stage of our research is to implement a static microcompiler and the run-time microengine (a better word might have been *microkernel*, but that term is taken). Our microengines execute microprograms, like the hardware microengines. The main technique used in this stage is *specialization*, used extensively in current operating system research, including Synthesis project at Columbia University, Synthetix project at Oregon Graduate Institute, Spin project at University of Washington, and Mach 4.0 project at University of Utah. Specialization has been shown to be very effective in the optimization of kernel calls. However, disciplined use of specialized kernel code has been a challenging problem. In our proposed research, the microcompiler generates code that directs the underlying kernel specialization, and the microengine generates the specialized code at run-time, if necessary, for best results.

The third stage is to write higher level software and evaluate the microlanguage fitness to the application area and the improvements achieved as compared to software without the microlanguage. In the past, specialization of operating system kernel calls have been evaluated based primarily on microbenchmarks, comparing kernel calls before optimization and after optimization. We have done the same in the Synthetix experiment on HP-UX [16].

However, microbenchmarks have their limitations. Specifically, they show only the effects on the particular kernel call, isolated from the rest of the system. In reality, it is the end-to-end performance that is of most interest to the user. Since microlanguages can facilitate cross-layer optimization, we will develop suitable benchmarks that will compare effectively the performance of microlanguage-based systems and those without microlanguages.

The fourth stage is to use the evaluation results to refine the microlanguage and its attendant tools such as microcompiler and microengine. Having done the experimental evaluation of systems with microlanguages, we will use the results to refine the experiment and the system. In particular, we will extend/reduce the microlanguage appropriately, according to the evaluation. Using our tools, we will then modify the microcompiler and microengine to re-evaluate our system to verify the impact of our refinements.

4 Technical Results

During the period covered by this report, we have made significant advances in our project. We first outline a selection of our publications and relevant papers written during this period.

4.1 Papers and Publications

During the Oct–Dec 1995 period:

1. Jonathan Walpole presented our paper [16] at the 1995 ACM Symposium on Operating Systems Principles (SOSP). The paper described our experience with applying specialization techniques systematically to a commercial operating system (HP-UX). In this paper, we have demonstrated the potential gains of specialization in performance and modularity on the one hand, and on the other hand, the difficulties with verifying the correctness of the specialized code, requiring powerful software tools for the large scale application of specialization. The SOSP is the bi-annual premier conference in operating systems research.

2. Shanwei Cen presented a demonstration [2] of our real-time distributed MPEG video/audio player at the 1995 ACM International Conference on Multimedia. The demo showcased our software-feedback based software, which has been freely available on the Web since April 1995.

3. Our paper [10] on "Adaptive Methods for Distributed Video Presentation" has appeared in a special Symposium on Multimedia of the ACM Computing Surveys journal. The paper analyzes the problems and solutions for delivering real-time multimedia presentations across the Internet, with special attention to our approach based on software feedback and our demonstration software.

During the Jan–Mar 1996 period:

1. Our paper [5] on "A General Approach for Run-Time Specialization and its Application to C" has been presented in the 1996 ACM Symposium on Principles on Programming Languages (POPL). This paper describes the theoretical foundation of the C on-line partial evaluator Consel is developing in France as part of the Synthetix specialization toolkit. The POPL is the annual premier conference on programming languages research.

2. Our paper [20] on "Safe Operating System Specialization: the RPC Case Study" has been presented at the First Workshop on Compiler Support for Systems Software (WCSSS), Arizona, February 1996. This paper describes the experiments in applying specialization to the RPC code of the Chorus microkernel.

3. Our paper [4] on "A Uniform Approach to Compile-Time and Run-Time Specialization" has been presented at the 1996 Dagstuhl Workshop on Partial Evaluation, Germany, February 1996. This paper describes the general approach of the Rennes group to specialization at both compile-time and run-time.

During the Apr–June 1996 period:

1. The paper [11] on "A Wait-free Algorithm for Optimistic Programming: HOPE Realized", by Crispin Cowan and Hanan L. Lutfiyya, was presented at the 1996 International Conference on Distributed Computing Systems, Hong Kong, May 1996.

2. Our paper [19] on "A Uniform Automatic Approach to Copy Elimination in System Extensions via Program Specialization", by Volanschi, E.-N., Muller, G., Consel, C., Hornof, L., Noye, J. and Pu, C., was published as IRISA Research Report No. 1021, Rennes, France, June 1996. It was submitted to the 1996 Symposium on Operating System Design and Implementation, Seattle, October 1996.

3. Our paper [7] "Automated Guarding Tools for Adaptive Operating Systems", by Crispin Cowan, Andrew Black, Charles Krasic, Calton Pu, and Jonathan Walpole was submitted to the 1996 Symposium on Operating System Design and Implementation, Seattle, October 1996.

4. Our paper [9] "Specialization Classes: An Object Framework for Specialization", by Crispin Cowan, Andrew Black, Charles Krasic, Calton Pu, and Jonathan Walpole of OGI, and Charles Consel and Eugen-Nicolae Volanschi of University of Rennes / IRISA, was submitted to the 1996 International Workshop on Object Orientation in Operating Systems, Seattle, October 1996.

Pending journal submissions:

1. We have submitted for journal publication an extended version of a recent conference paper [12]. The paper describes the formal semantics for expressing optimism, which is a general approach for parallel and distributed computation compatible with specialization.

2. We have been invited to submit our paper "Customizable Operating Systems" as a book chapter, being edited as a result of a special issue of CACM on advances on operating systems.

4.2 Presentations in This Period

1. Andrew Black presented the overview of the project at the University of Glasgow in a visit during March 1996.

2. Adaptive Operating Systems (overview of Synthetix in the large) presented at the University of Waterloo department of Computer Science on June 20th.

3. Adaptive Real-time Scheduling for Multimedia Quality of Service in a UNIX Environment. Guest lecture to the University of Waterloo department of Computer Science CS 452 (Real Time Systems) course. Presented on June 21st.

4.3 Contract Deliverables

4.3.1 Oct–Dec 1995

Specialization Methodology We have been refining the specialization methodology in our work. A preliminary version of the methodology has been described in our SOSP paper [16]. The refined methodology, which includes the use of and support for microlanguages, is under active development. This is both Synthetix and Microlanguage deliverable.

Specialization Toolkit We have been designing and implementing the specialization toolkit during this period. Our main accomplishment is the creation and refinement of the concept of *specialization class*. Each concrete specialization is an instance of a general type of specialization, defined by the specialization class. Many tools in the specialization toolkit are based on this concept. The explicit (and machine readable) definition of specialization classes allows the software tools to communicate with each other easily. We are writing several papers on specialization classes and developing code using them. Preliminary versions of specific tools have been under test by the members of our group, including the Tempo-C on-line partial evaluator in France and the dynamic kernel module replugger at OGI. The toolkit includes deliverables from both the Synthetix grant and the Microlanguages contract.

Software Feedback We have been designing and implementing a software feedback toolkit for fine-grain adaptive distributed computation. There are two recent developments in this effort. First, conceptually, we are modeling software feedback as a fine-grain specialization technique. This way, we will be able to use the specialization tools to support software feedback, and integrate the software feedback toolkit into the specialization toolkit easily. Second, concretely, we are re-implementing the distributed multimedia video/audio player to use the software feedback toolkit. This is a Synthetix deliverable.

Microlanguage Development We have started the development of our first microlanguage for file systems. From the conceptual side, this effort is based on recent work on the specialization of file systems such as our SOSP paper [16] and other research papers on choosing the appropriate caching strategy (e.g., [13]) for an application. From the systems building side, this work is integrated with the construction of the specialization toolkit. We are re-designing the specialization tools to make them extensible and add hooks supporting microlanguages from the beginning. This is a Microlanguages deliverable.

4.3.2 Jan–Mar 1996

Specialization Methodology We have been refining the specialization methodology in our work. In this quarter, we have been designing the next generation specialization experiments in the context of Quality of Service support for multimedia and dynamic reconfiguration of mobile networks. This is both a Synthetix and Microlanguage deliverable.

Specialization Toolkit We have been refining the concept of specialization class. Each concrete specialization is an instance of a general type of specialization, defined by the specialization class. Currently, the specialization class is considered the "assembly language" of specialization, into which the microprograms will be translated. Components completed for alpha testing include the Replugger [CDS96], the type-based guard checker, and the preliminary version of the Tempo-C compiler [POPL96]. The memory run-time guard checker is under development at OGI. The specialization toolkit will include deliverables from both the Synthetix grant and the Microlanguages contract.

Software Feedback We have redesigned the software feedback toolkit. Currently, it consists of a component library being written in C++, composition tools, and guarding tools built using the specialization toolkit. The component library will be illustrated using a new demonstration program that has enhanced functionality compared to the distributed multimedia MPEG player we released in May 1995. This is a Synthetix deliverable.

Microlanguage Development We have started the development of our first family of microlanguages. The family is an integrated set of four microlanguages to describe and program multimedia applications. At the top level, it will be a Quality of Service (QoS) specification microlanguage, based on Richard Staehli's dissertation work [18] under Jonathan Walpole. The QoS specification is then translated into a media-aware scheduling microlanguage, which knows about the compression method, time binding, frame dependencies, and other resource management issues. The data access part of the scheduling microlanguage is then translated into a low level file system microlanguage, which knows about the details of file representation and access. The file system microlanguage is finally translated into the specialization classes being written for the kernel experiments, including read-ahead, buffer cache usage and management, and I/O system performance. This is a Microlanguages deliverable.

Experiments Using Microlanguages We have not started the experimentation of systems with microlanguages, which is dependent on the design and implementation of microlanguages. This is a Microlanguages deliverable.

4.3.3 Apr–June 1996

Specialization Methodology We have been refining the specialization methodology in our work. Using the plans for experimentation, we have been designing and implementing the specialization and software feedback toolkit as described below. A new experiment on

automated elimination of copying has been reported [19]. This is both a Synthetix and Microlanguage deliverable.

Specialization Toolkit We have been refining the concept of specialization class. In this quarter, we have completed two new papers. The first paper [7] describes the first part of the toolkit. The second paper [9] describes the current refinements of specialization classes. Besides the Replugger [7], the type-based guard checker, and the preliminary version of the Tempo-C compiler [5], we have continued the development of the memory run-time guard checker. The specialization toolkit will include deliverables from both the Synthetix grant and the Microlanguages contract.

Software Feedback We have been implementing the redesigned the software feedback toolkit. Currently, it consists of a component library being written in C++, composition tools, and guarding tools built using the specialization toolkit. The component library will be illustrated using a new demonstration program that has enhanced functionality compared to the distributed multimedia MPEG player we released in May 1995. This is a Synthetix deliverable.

Microlanguage Development We have continued the development of our first family of microlanguages. The design has been refined using our experience with the implementations of the multimedia player and the file system specialization experiment. The family is an integrated set of four microlanguages to describe and program multimedia applications. At the top level, it will be a Quality of Service (QoS) specification microlanguage. The QoS specification is then translated into a media-aware scheduling microlanguage. The data access part of the scheduling microlanguage is then translated into a low level file system microlanguage. The file system microlanguage is finally translated into specialization classes. Mr. Thimm is working on the high-level microlanguage for Quality of Service specification and software integration of Synthetix player and GMD/IPSI's multimedia storage support programs. This is a Microlanguages deliverable.

5 Important Findings

The Microlanguages contract consists of a combination of efforts from the operating systems side and from the programming languages side. We have summarized our important findings in two papers, both appearing in the most important conference of the area. In our SOSP paper [16], our most important findings in operating systems are:

1. Systematic specialization is feasible and worthwhile (significant performance gains in the kernel calls studied). Our current work on specialization classes [9] follows this thread.

2. Large-scale specialization requires sophisticated and easy-to-use software tools. See the discussion below.

In our POPL paper [5], our most important findings in the programming languages area are:

1. It's possible to build an on-line partial evaluator (Tempo-C) to support specialization.
2. It's possible to apply the partial evaluator to significant pieces of operating system code written in (cleaned-up) C. Our current experiments [19] follow this thread.

Our experimental work [4, 20, 10, 2] has validated our research ideas and provided directions to its continued development.

Our toolkit construction work is making good progress. The specialization toolkit consists of alpha test of the Replugger [7], Tempo-C compiler [5], and type-based guard checker [8], with the memory run-time guard checker being integrated into the Linux kernel. The software feedback toolkit consists of a alpha test of the component library and a re-implementation of the distributed multimedia MPEG player [3].

6 Significant Hardware Developments

The Synthetix project is a software development project. We have not planned any hardware developments. During this period of contract performance, no unexpected hardware developments were necessary.

7 Special Comments on Industrial Collaboration

In the Synthetix project, we have maintained an active collaboration with industry. Currently, we have the following active collaborations:

- Tektronix: distributed multimedia application support based on software feedback. We are receiving funding from Tektronix for this collaboration.
- Hewlett-Packard: specialization experiments on HP-UX and specialization toolkit development. This collaboration is with both the HP-UX maintenance group and HP Labs operating systems research groups. We received funding in 1993–1994 and with continued collaboration we expect HP to resume funding.
- Xerox PARC: use of reflection in specialization and the application of specialization as reflection.
- Intel: application of specialization to distributed multimedia and architectural support for specialization. We received funding in 1996 for this collaboration.

8 Implication for Further Research

The project is ongoing and we are working on the following aspects of the research:

- Specialization Methodology: continued refinement of the methodology, especially in conjunction with the development of microlanguages. Specialization class is a concept that ties together the specialization methodology and tools in the specialization toolkit.
- Specialization Toolkit: continued development of specialization tools. We are in alpha testing of the dynamic kernel replugger and the Tempo-C partial evaluator. At the same time, we are finishing the development of type-based guard checker, a program to point out all the physical locations where program code touches on a memory location. We are also working on a hardware-based memory run-time guard checker, to protect the integrity of specialization invariants at run-time. Specialization classes are the interfaces among tools.
- Software Feedback: continued development of the software feedback toolkit as well as the demonstration program.
- Microlanguage development: continued design and implementation of microlanguages. The first family of microlanguages consists of a Quality of Service specification microlanguage, a resource scheduling microlanguage, a file system microlanguage, and appropriate specialization classes for the implementation of multimedia presentations over specialized operating systems.

A Attached Papers

Two papers are attached to this technical report.

1. Paper in 1996 IEEE ICDCS [11]. Labeled Appendix B.
2. Paper submitted to the 1996 IWOOOS [9]. Labeled Appendix C.

References

[1] B.N. Bershad, C. Chambers, S. Eggers, C. Maeda, D. McNamee, P. Pardyak, S. Savage, and E.G. Sirer. SPIN - An Extensible Microkernel for Application-specific Operating System Services. In *SIGOPS 1994 European Workshop*, February 1994. UW Technical Report 94-03-03.

[2] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. Demonstrating the effect of software feedback on a distributed real-time MPEG video audio player. In *Proceedings of the 1995 ACM International Conference on Multimedia*, San Francisco, November 1995.

[3] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A distributed real-time MPEG video audio player. In Tom Little, editor, *Proceedings of the 1995 International Workshop on Network and Operating System Support for Digital Audio and Video*, volume 1018 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995. Also appeared in the Proceedings of the 1995 International Workshop on Network and Operating System Support for Digital Audio and Video, April 1995, New Hampshire.

[4] C. Consel, L. Hornof, F. Noel, and E.N. Volanschi. A uniform approach to compile-time and run-time specialization. In *Proceedings of the 1996 Dagstuhl Workshop on Partial Evaluation*, Germany, February 1996.

[5] C. Consel and F. Noel. A general approach for run-time specialization and its application to c. In *Proceedings of the 1996 ACM Symposium on Principles of Programming Languages*, Florida, January 1996.

[6] C. Consel, C. Pu, and J. Walpole. Incremental partial evaluation: The key to high performance, modularity and portability in operating systems. In *Proceedings of the 1993 ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, Copenhagen, Denmark, June 1993.

[7] C. Cowan, T. Autrey, C. Krasic, C. Pu, and J. Walpole. Fast concurrent dynamic linking for an adaptive operating system. In *Proceedings of the International Conference on Configurable Distributed Systems*, Maryland, May 1996.

[8] C. Cowan, A. Black, C. Krasic, C. Pu, and J. Walpole. Automated guarding tools for adaptive operating systems. Technical Report OGI-CSE-95-0XX, Department of Computer Science and Engineering, Oregon Graduate Institute, May 1996.

[9] C. Cowan, A. Black, C. Krasic, C. Pu, J. Walpole, C. Consel, and E.-N. Volanschi. Specialization classes: An object framework for specialization. Technical Report OGI-CSE-95-0YY, Department of Computer Science and Engineering, Oregon Graduate Institute, July 1996.

[10] C. Cowan, S. Cen, J. Walpole, and C. Pu. Adaptive methods for distributed video presentation. *ACM Computing Surveys*, 27, December 1995.

[11] C. Cowan and H. Lutfiyya. A wait-free algorithm for optimistic programming: Hope realized. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, Hong Kong, May 1996.

[12] Crispin Cowan and Hanan Lutfiyya. Formal Semantics for Expressing Optimism: The Meaning of HOPE. In *1995 Symposium on the Principles of Distributed Computing (PODC)*, pages 164–173, Ottawa, Ontario, August 1995.

[13] D. Kotz. Disk-oriented I/O for MIMD multiprocessors. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 61–74, Monterrey, CA, November 1994. Usenix.

[14] H. Massalin and C. Pu. Threads and input/output in the Synthesis kernel. In *Proceedings of the Twelfth ACM Symposium on Operating System Principles*, pages 191–201, Arizona, December 1989.

[15] S. O'Malley and L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110–143, May 1992.

[16] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang. Optimistic incremental specialization: Streamlining a commercial operating system. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, Colorado, December 1995.

[17] C. Pu, H. Massalin, and J. Ioannidis. The Synthesis kernel. *Computing Systems*, 1(1):11–32, Winter 1988.

[18] R. Staehli. *Quality of Service Specifications for Resource Management of Multimedia Systems.* PhD thesis, Department of Computer Science and Engineering, Oregon Graduate Institute of Science & Technology, 1995.

[19] E.-N. Volanschi, G. Muller, C. Consel, L. Hornof, J. Noyé, and C. Pu. A uniform automatic approach to copy elimination in system extensions via program specialization. Research Report 1021, Irisa, Rennes, France, June 1996.

[20] E.N. Volanschi, G. Muller, and C. Consel. Safe operating system specialization: the rpc case study. In *Proceedings of the First Workshop on Compiler Support for Systems Software*, Arizona, February 1996.