

Microsoft®

Research

Phoenix Academic Program



PHOENIX

CODEGEN . OPTIMIZATION . ANALYSIS

External Research & Programs

<http://research.microsoft.com/Phoenix>

ABOUT THIS REPORT

This report celebrates the accomplishments of the Microsoft Phoenix Academic Program.

Over the last four years, we've been fortunate to have the opportunity to work with outstanding professors and students on using Microsoft Phoenix as a compiler and application development framework. Our collaborations have resulted in creating innovative solutions for computer science research and education. You will find program highlights in this booklet and detailed information in accompanying CD-ROM.

Please contact us, if you have any questions, feedback or comments regarding the Microsoft Phoenix Academic Program.

Dr. Yan Xu
External Research & Programs
Microsoft Research
<http://research.microsoft.com/Phoenix/>

July, 2007

CONTENTS

BACKGROUND	4
RESEARCH HIGHLIGHTS	5
Dr. Mary Lou Soffa	5
Developing a Testing Framework for Security.....	6
Refining Buffer Overflow Detection via Demand-Driven Path-Sensitive Analysis.....	8
Dr. Jack Davidson	10
Teaching Computer Security Using Phoenix.....	11
Dr. Wen-mei Hwu	14
Prof. Dr. Andreas Polze	15
Phoenix to Support Domain-specific Languages in the Distributed Control Lab	15
Phoenix for .NET Aspect Weaving.....	16
Phoenix for Embedded Systems Code Optimization.....	16
Phoenix for Dynamic Windows Research Kernel Code Visualization	17
Dr. Alfred Aho	18
Debugging Aspect-Enabled Programs.....	19
WICCA by Marc Eaddy	20
Phx.Morph by Marc Eaddy	21
Dr. Regeti Govindarajulu	22
Phoenix-Based Compiler Course	22
Professor Vladimir Safonov	24
High-Level Language Abstract Syntax Tree For Phoenix.....	25
Phoenix Front-End Toolkit and Environment.....	26
Dr. Eric Wohlstadter	27
Dr. Kris De Volder	27
Dr. Chen Ding	28
Waste Not,Want Not: Adaptive Garbage Collection in a Shared Environment.....	28
MICROSOFT PHOENIX RESEARCH AWARDS	30
Phoenix Early Adopter Awards – 2004	30
Phoenix RFP Awards – 2005	31
Phoenix and SSCLI Awards – 2006	35
Phoenix Direct Funding Awards – 2007	39
PHOENIX DOWNLOADS	40
OTHER RESOURCES	41
Phoenix Users Forum	41

BACKGROUND

Microsoft Phoenix

Phoenix is the code name for a software optimization and analysis framework that is the basis for all future Microsoft compiler technologies. The Phoenix framework is an extensible system that can be adapted to read and write binaries and Microsoft Intermediate Language (MSIL) assemblies and represent the input files in an IR, which can be analyzed and manipulated by applications by using the Phoenix API. The code can then be written in binary or JITable form for execution.

Microsoft Phoenix Academic Program

Microsoft Research launched the Microsoft Phoenix Academic Program in 2003 to provide researchers early access to the latest compiler technology and to obtain feedback from domain experts through research collaboration. The program allowed Microsoft to assess and improve the value of Phoenix for the research and instructional communities.

Microsoft Phoenix Research Development Kit (RDK)

The Phoenix RDK is the core of the Phoenix Academic Program. It contains the Microsoft Phoenix framework, a group of samples and API reference documentation. In September of 2003, as part of the Phoenix Early Adopter Awards, the RDK was released to four principal investigators for their university research projects. Since then, the RDK has been updated and released two or three times per year, with the last one in March 2007.

Microsoft Phoenix Research

Over the last four years, the Phoenix Academic Program has sponsored over forty research projects with over \$1.7 million in total funding. External research started in 2003 with the Phoenix Early Adopter Awards and was followed in 2005 with the Phoenix RFP Awards. In 2006, eighteen projects were sponsored with the Phoenix and SCLI Awards. The 2007 Phoenix Direct Funding Awards brought the final four projects into the program.

What's Next

Later this year, Microsoft will release the first Phoenix Software Development Kit (SDK). This will end Phoenix RDK development and it will open new opportunities for Phoenix-based collaboration at the next level.

RESEARCH HIGHLIGHTS

Over forty researchers have greatly contributed to the Microsoft Phoenix Academic Program. The following ten are highlighted to show a sample of the variety and the depth of sponsored research. These highlights are selected extracts, for full documents or additional information see accompanying CD-ROM.



DR. MARY LOU SOFFA

Owen T. Cheatham Professor of Sciences and Department Chair
Department of Computer Science
University of Virginia (<http://www.virginia.edu/>)
Charlottesville, Virginia

Email: soffa@cs.virginia.edu
Home page: <http://www.cs.virginia.edu/~soffa/>

Mary Lou Sofa received her B.S. and M.S. in Mathematics and her Ph.D. in Computer Science. From 1977 to 2004, she was a Professor of Computer Science at the University of Pittsburgh and served as the Dean of Graduate Studies in the College of Arts and Sciences from 1991 to 1996. In 2004, she moved to the Department of Computer Science at the University of Virginia, where is the Owen T. Cheatham Professor and Department Chair of the Computer Science Department.

She received the Nico Habermann Award in 2006 for outstanding contributions toward increasing the numbers and successes of underrepresented members in the computing research community. In 1999, she received the Whitehouse's Presidential Award for Excellence in Science, Mathematics and Engineering Mentoring. She was elected an ACM Fellow in 1999 and selected as a Girl Scout Woman of Distinction in 2003. She served for ten years on the Board of the Computing Research Association (CRA) and continues as a member of CRA-W, the committee on the status of women in computer science and engineering of the CRA. She has served on the Executive Committees of both ACM SIGSOFT and SIGPLAN as well as conference chair, program chair or program committee member of many conferences. Currently, she is the program Conference Chair for the Code Generation and Optimization Conference (CGO). She has been a distinguished speaker at a number of conferences and universities including the Fifth International Conference on Quality Software, Compiler Construction Conference, Static Analysis Symposium, University of Illinois, the University of Maryland, Notre Dame, Stony Brook, and the University of Michigan.

Her research interests include software tools for debugging and testing programs, compilers, optimizations and program analysis. She has published over 140 papers in journals and conferences. Her papers have received a number of best paper awards as well a designation of one of the 40 most influential papers in 20 years to appear in the Programming Language Design and Implementation Conference. She has directed 25 Ph.D. students to completion, half of whom are women. Her former Ph.D. students are professors in major universities including the University of California at Berkeley, Georgia Tech, the University of Maryland, the University of Arizona and the University of Delaware.

Dr. Mary Lou Sofa received a Phoenix and SCLI award in 2006 for her project *Developing a Testing Framework for Security*. Most recently, she received a Phoenix Direct Funding award for her research project *Software Testing for Security Vulnerabilities*. Read about Dr. Sofa's projects below and in accompanying CD-ROM.

DEVELOPING A TESTING FRAMEWORK FOR SECURITY

Dr. Mary Lou Soffa
Department of Computer Science
University of Virginia

Introduction

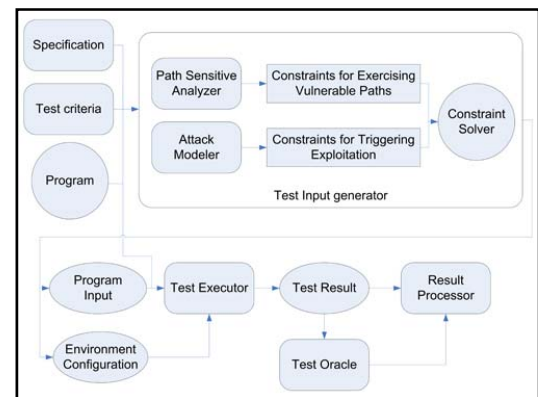
Despite increasing efforts in detecting and managing software security flaws, the number of security attacks is still rising every year [5, 21]. As software becomes more complex, security flaws are more easily introduced into a system and more difficult to eliminate. Meanwhile, attackers take advantage of software tools for developing, testing and debugging exploits, which make the construction of attacks automatic, fast and easy [20].

There have been many approaches to the challenges of managing security flaws, including solutions that monitor and protect systems after software is deployed, e.g., intrusion detection and runtime monitoring. These tools are all reactive, in that they detect vulnerabilities when attacks occur. Simply rebooting for recovery may lead to denial of service for the software system. A safer approach is patching software, which is expensive and needs to be done faster than the spread of the malware. There have also been efforts to detect and eliminate security flaws before software release, including static analysis, model checking and testing approaches. The challenge for static tools is the balance between precision and scalability since either superfluous false positives or high overhead will make tools useless in practice. Testing for security flaws has not received much research attention. Existing approaches include random-testing based Fuzz and instrumentation based buffer overflow testing [10, 12]. However, they both report low coverage of the code. Manual penetration testing is still widely used, but it is not scalable.

Testing for correctness of the software detects inconsistencies between software behavior and functional specification or by crashing the program. Similarly, testing could be used to detect security flaws by demonstrating inconsistencies between software behavior and security policies or symptoms of violating security coding rules. After all, security flaws are a special type of design or coding flaws. Testing can play an important part in producing software systems that are secure just as it plays a major role in helping to guarantee the correctness of programs.

Despite the similarity between security flaws and flaws that prevent programs from functioning correctly, security flaws have the feature of being hard to detect after being implanted into programs, making it impossible to check them with traditional correctness-checking tools. Applying existing testing techniques to check for security violations involves several challenges. Firstly, according to the cause of the vulnerabilities, security flaws are often located at exceptional control flow branches, triggered only when a boundary condition is satisfied (e.g., flaws cause integer overflow) and exposed only by a very special input. It is difficult to model vulnerabilities that describe these control and data conditions properly. Finite state machines, formal specifications, coding signatures or their combination are candidates for representing these conditions. Secondly, in order to check programs against vulnerabilities efficiently, an intermediate representation of the program that represents security control and data flow of certain types of vulnerability needs to be developed. The third difficulty is to determine the environmental factors that trigger vulnerabilities and the ways to simulate them in testing.

Figure 1: Security Testing Framework



The goal of this research project is to develop a testing framework for detecting and managing security flaws. The key idea is to develop static analysis tools to determine program paths that lead to various types of vulnerabilities, and then apply testing techniques to exploit the vulnerabilities. The static analysis algorithms will be path-sensitive to provide detailed information about the security flaw. They will also be demand-driven to ensure scalability to handle large programs. Test inputs will be automatically generated to exercise potentially vulnerable paths and paths where a determination of whether vulnerability exists cannot be made statically to try to demonstrate vulnerability. By demonstrating real exploits through testing, we aim to eliminate (or at least reduce) false alarms generated from static analysis. We plan to consider several popular vulnerabilities: memory safety, denial of service caused by exhausted resources, privilege escalation and resource access control violations [5, 18, 21].

Use of Microsoft Phoenix

In this project, Microsoft Phoenix [24] is being used to implement the static analysis tools to compute potentially vulnerable paths for software. More specifically, Phoenix APIs are helping in the construction of an intermediate representation of the software, and then used to build the demand-driven data flow analysis to collect information through source code.

Expected Contributions

The primary goal of our research is to develop robust, flexible, and scalable techniques for determining security flaws before software release. The expected contributions include:

- A framework for testing software for security flaws. It will include demand-driven path-sensitive analyses as well as testing components including a test input generator, a set of criteria, an executor, an oracle and a result processor. The framework will be written with the support of Microsoft's Phoenix, and tested with buffer overflow benchmarks, BugBench, SSCIL and the Windows Research Kernel.
- Test criteria and a test coverage tool for testers to judge the quality of their security test suite.
- A security knowledge base that documents and analyzes a set of software vulnerabilities and attacks.
- A set of synthetic test inputs for Windows applications such as SSCIL and the Windows Research Kernel for gauging the effectiveness of our techniques and those of other researchers at countering attacks.
- A set of APIs for tester developers to build tools for detecting their own defined vulnerabilities.

More Information

For a complete project whitepaper and references, see accompanying CD-ROM.

References

[5] CERT: <http://www.cert.org/>

[10] J. E. Forrester and B. P. Miller, An Empirical Study of the Robustness of Windows NT Applications Using Random Testing, 4th USENIX Windows Systems Symposium, 2000.

[12] E. Haugh and M. Bishop, Testing C Programs for Buffer Overflow Vulnerabilities, Proceedings of the Network and Distributed System Security Symposium, 2003.

[18] G. Hoglund and G. McGraw, Exploiting Software, Addison-Wesley Professional, 2004.

[20] The Metasploit Project: <http://www.metasploit.com/>

[21] National Vulnerability Database: <http://nvd.nist.gov/>

[24] Phoenix: <http://research.microsoft.com/phoenix/>

REFINING BUFFER OVERFLOW DETECTION VIA DEMAND-DRIVEN PATH-SENSITIVE ANALYSIS

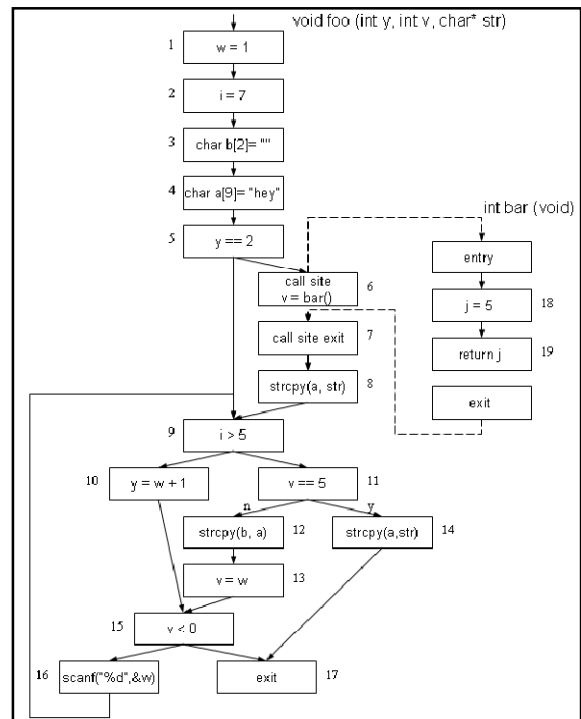
Wei Le and Mary Lou Soffa
Department of Computer Science
University of Virginia
{weile, soffa}@cs.virginia.edu

Introduction

Although much effort has been expended to detect and avoid buffer overflow in software, we are still plagued with exploits that are costly to fix, disruptive, and promote a general loss of trust in software. Since many applications are written in unsafe languages and it is difficult for programmers to correctly write applications that use buffers, buffer overflow is still being introduced into software and is the most commonly exploited vulnerability [5, 21]. In 2006, SecuriTeam reported 134 vulnerable overflows, a quarter of the total security warnings [21], and many of them have caused severe impact such as unauthorized access and denial of service. To detect vulnerabilities, dynamic detectors are used but they slow down the execution by a factor of 2 to 30 due to the increase of code size, branch mispredictions and data cache misses [24]. Therefore dynamic buffer overflow detection is difficult to apply for time constrained software. In addition, patches to fix the vulnerability are expensive due to the number of computers typically affected. For these reasons, a number of software companies rely on static analysis to detect buffer overflow before software release [12,13].

However, current static tools require considerable human effort, either for diagnosing warnings or for annotating programs to help analysis [4, 11, 13, 18, 22, 24]. Many tools report warnings about potentially vulnerable program points, such as statements or buffers, for example, Splint, BOON and ARCHER [11, 22, 24]. The code reviewer has no knowledge about the paths through the program point that actually produce the vulnerability. Tools that report vulnerable paths instead of statements include Prefix, ESPx and Prefast [4, 13, 18]. The analysis is performed exhaustively along all program paths. The challenge for these tools is scalability, in particular when the vulnerability may cross procedure boundaries. As a result, the tools sometime have to give up after exploring a certain number of paths [18]. Although heuristics can be applied to select and merge paths, excessive warnings are produced [4]. Some tools address scalability by introducing annotations to specify the buffer contract between procedures and thus turn the buffer overflow detection intraprocedural [13]. But both writing and verification of annotations are costly, and thus, correctness of annotations is not guaranteed.

Figure 2: A Simple Example



This work develops an interprocedural demand-driven path-sensitive analysis with the goal of reducing the effort required to identify program paths that are vulnerable and providing more precise information about the vulnerability to help users find the root cause. Our analysis classifies paths as

infeasible, safe, vulnerable with potential for exploits, overflow with little chance to be exploited, and don't-know. Our analysis is driven by statements that have a definition or redefinition of a buffer. By using a demand driven algorithm, our analysis is directed to those paths that can be executed and maybe vulnerable, and the analysis terminates as soon as the vulnerability decision is discovered.

Through our analysis, we exclude paths that are infeasible and safe, and prioritize paths that can overflow based on their chance of being exploited.

Contributions

- A categorization and identification of five types of paths for buffer overflow.
- An interprocedural demand-driven path-sensitive diagnosis tool for identifying the types of paths through a potential overflow buffer.
- Experimental results that demonstrate the path types existing in real programs and the time and space costs of the analysis.

More Information

For the complete whitepaper and references, see accompanying CD-ROM.

References

- [4] W. R. Bush, J. D. Pincus, and D. J. Siela. A static analyzer for finding dynamic programming errors. *Software: Practice and Experience*, 2000.
- [5] CERT. <http://www.cert.org>.
- [11] D. Evans. Static detection of dynamic memory errors. In *Proceedings of the ACM SIGPLAN 1996 Conference on Programming language Design and Implementation*, 1996.
- [12] Fortify. <http://www.fortifysoftware.com>.
- [13] B. Hackett, M. Das, D. Wang, and Z. Yang. Modular checking for buffer overflows in the large. In *Proceedings of the 28th international Conference on Software Engineering*, 2006.
- [18] Microsoft. Prefast. <http://www.microsoft.com/whdc/devtools/tools/prefast.msp>.
- [21] SecuriTeam. <http://www.securiteam.com/>.
- [22] D. Wagner, J. S. Foster, E. A. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In *Proceedings of Network and Distributed System Security Symposium*, 2000.
- [24] Y. Xie, A. Chou, and D. Engler. ARCHER: Using symbolic, path-sensitive analysis to detect memory access errors. In *Proceedings of the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2003.



DR. JACK DAVIDSON

Professor of Computer Science
University of Virginia (<http://www.virginia.edu/>)
Charlottesville, Virginia

Email: davidson@cs.virginia.edu

Home page: <http://www.cs.virginia.edu/brochure/profs/davidson.html/>

Jack Davidson received his Ph.D. in Computer Science at the University of Arizona in 1981. He joined UVa as an Assistant Professor of Computer Science in 1982, becoming Associate Professor in 1988 and Professor in 1998. In 1997 he received the McGraw-Hill "Most Successful New title" Award for his best-selling C++ textbook (co-authored with Jim Cohoon). His more recent book *Java Program Design* was published in 2003. He is Associate Editor for ACM's *Transactions on Architecture and Code Optimization*. He has directed eight Ph.D. theses and is the co-author of two books and over 120 technical articles.

Dr. Davidson's research focuses on two complementary areas of computer science: compiler construction and computer architecture. Since the performance of a computer system depends on interaction between the hardware and software, little advantage is gained by including architectural enhancements that the compiler cannot exploit. Davidson's research investigates this interaction with a goal of developing effective solutions. In compiler construction, he investigates the development of easily retargetable, highly optimizing compilers. Earlier research developed an intermediate representation, RTL, which is the basis for two widely distributed and widely used retargetable, optimizing compilers, the GNU C compiler and vpo. He was a principal investigator of the National Compiler Infrastructure (NCI) project, which developed Zephyr, a tool suite for compiler and architecture research. Currently he is designing and building new software development environments for high-performance embedded applications (e.g., wireless video, digital cameras, etc.). He is also working on dynamic optimization with a focus software security.

Dr. Davidson received a Phoenix RFP Award in 2005 for his project *Techniques and Tools for Software Assurance* and a Phoenix and SCLI Award in 2006 for *Using Phoenix in Anti-Virus Curricula* project. Read more about Dr. Davidson's projects below. For the complete whitepaper, see accompanying CD-ROM.

TEACHING COMPUTER SECURITY USING PHOENIX

Mark Bailey
Hamilton College
mbailey@hamilton.edu

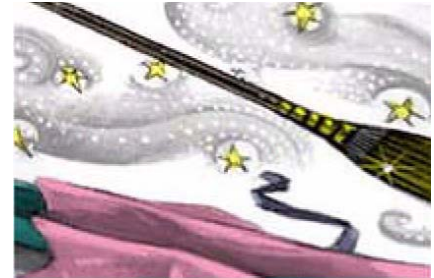
Clark Coleman and Jack Davidson
University of Virginia
{clc5q,jwd}@virginia.edu

Jeff Zadeh
Virginia State University
jzadeh@vsu.edu

Overview and Motivation

Increasing security threats motivate the development of security courses to ensure that computer science graduates are prepared for future challenges. Our graduates should know how to detect, prevent and defeat viruses and security exploits. In developing a course on anti-virus concepts, we discovered that core computer science concepts that often taught in a theoretical manner can be taught within the interesting application area of computer security. The course, *Defense against the Dark Arts*, was first taught in the fall semester of 2005 at the University of Virginia. The class web pages had a Harry Potter design theme (see graphic on the right). The theme, course title, and subject matter helped fill the course enrollment limit of 50 students, well before the semester began.

Figure 3: Defense against the Dark Arts



Early Experience and Student Audience

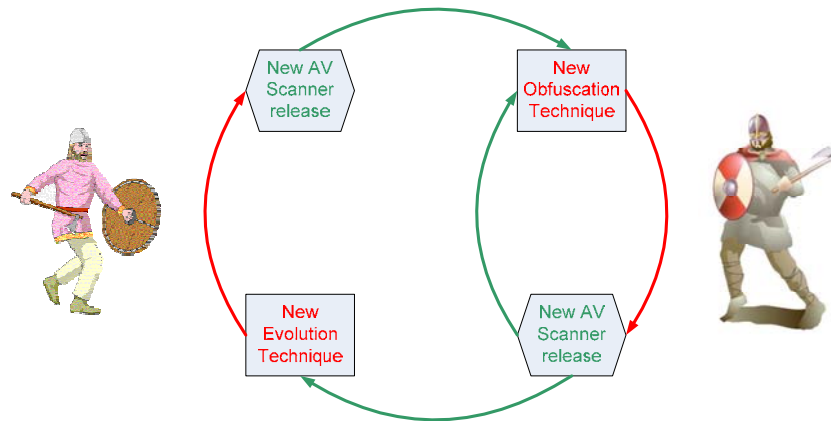
The course was refined and taught a second time in the spring of 2007, at both the University of Virginia and Hamilton College. Curriculum ideas from the course were integrated into an existing security course at Virginia State University, an HBCU. Vetting the course at three such differing institutions is good preparation for disseminating the course materials in the near future through the Microsoft Academic Alliance Curriculum Repository.

Both experiences at Virginia attracted mostly third and fourth year computer science majors. In the spring of 2007, about a dozen non-majors enrolled; half were computer engineering majors and two students were from outside the engineering school. The course has attracted larger female student enrollment than comparable upper-level elective courses (6 female students out of 34 in 2007 at Virginia).

The Ongoing Battle

Students are taught some of the history of virus and anti-virus software to impress upon them the nature of the battle between virus authors and anti-virus researchers. As each new generation of anti-virus software has been released, virus authors have designed new techniques to evade detection. Students are taught that there will never be a final victory in this battle, and that there are theoretical reasons why this is the case, as well as theoretical reasons that anti-virus software needs to be radically re-designed in the near future. Student interest is aroused by being part of an ongoing war.

Figure 4: The ongoing battle between virus writers and anti-virus researchers.



Core Computer Science Concepts

In many classes, core computer science concepts are taught in a purely theoretical context that is not interesting or motivating to students. Many core computer science concepts arise naturally in the teaching of an anti-virus course. For example, anti-virus scanners use pattern matching tools based on regular expressions. This provides an opportunity to teach regular expressions, finite state automata, and pattern matching tools such as *lex* or *flex*.

However, obfuscations make it impossible for regular expressions to detect viruses. Virus obfuscation shows the need for more powerful tools that can remember context and perform analyses on the control flow and data dependences in the code. The inadequacy of regular expressions to detect obfuscated or evolving viruses provides a good framework for introducing and discussing the Chomsky Hierarchy.

The problem of determining whether a given program is an obfuscated version of a known virus program is equivalent to solving the halting problem, another core concept that is usually taught without practical application.

These core concepts of theoretical computer science, coupled with programming assignments, convince the students that tools from higher levels of the Chomsky Hierarchy are required to reverse code obfuscations as used by virus authors. Programming assignments require students to reverse the effects of at least two common code obfuscations.

Obfuscations change virus code signatures without changing virus functionality. This signature change foils regular expression based virus scanners

Sophisticated obfuscations can be reversed by standard compiler analyses, for example:

- 1) Junk assignment insertion: Insert an assignment statement that changes a register value, but the new register value is not used by later instructions.

Before obfuscation

```
add eax,4  
sub edx,eax
```

After obfuscation

```
add eax,4  
add esi,7 ; register esi is never used with this value  
sub edx,eax
```

This obfuscation is easily detected by data flow analysis over the SSA (Static Single Assignment) form of the program, but not recognizable by regular expressions.

This example provides an opportunity to introduce SSA form and simple data flow analysis outside of a compiler course.

- 2) Adding unnecessary jumps: Transforms straight line code into spaghetti code with the same functionality:

<i>Before obfuscation</i>	<i>After obfuscation</i>
x += 4;	x += 4;
y += (z - x);	goto lab2;
z -= 3;	lab3: z -= 3;
printf("%d\n", x)	goto lab4;
	lab2: y += (z - x);
	goto lab3;
	lab4: printf("%d\n", x);

This obfuscation is easily reversed by the jump straightening transformation commonly found in optimizing compilers. This example enables teaching about control flow graphs outside of a compiler or programming languages course.

These programming assignments demonstrate the power of compiler analyses as compared to pattern matching tools. In particular, the students learn to use the power of the Phoenix compiler suite from Microsoft Research.

The Phoenix Research Compiler

The primary tool used for programming assignments is the Phoenix compiler suite from Microsoft Research. After an introductory assignment to introduce students to the Phoenix IR (intermediate representation) and gives them experience in creating a Phoenix analysis tool using a Visual Studio wizard, the next two assignments are realistic de-obfuscation transformations. The students have enjoyed receiving hands-on experience in the battle against virus creators. They selected Phoenix overwhelmingly over pattern matching tools as their favorite technology used in the course, according to anonymous surveys conducted at the end of the semester in 2007 at Virginia.

Not only does Phoenix provide built-in compiler analyses that are easy to use via tools and plugins, but it enhances the realism of anti-virus work by permitting the analysis of binary executables. Most compiler suites require the availability of source code, which is obviously not realistic in an anti-virus scanning environment. This enhanced realism was a primary factor in the positive overall evaluations of the course in anonymous end of semester surveys.

More Information

For the complete whitepaper, see accompanying CD-ROM.



DR. WEN-MEI HWU

AMD Jerry Sanders Chair of Electrical and Computer Engineering
University of Illinois-Urbana-Champaign (<http://www.uiuc.edu/>)
Urbana, Illinois

Email: hwu@crhc.uiuc.edu

Home page: <http://www.crhc.uiuc.edu/impact/people/current/hwu.php>

Wen-mei W. Hwu is the Walter J. ("Jerry") Sanders III-Advanced Micro Devices Endowed Chair in Electrical and Computer Engineering in the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign. From 1997 to 1999, Dr. Hwu served as the chairman of the Computer Engineering Program at the University of Illinois. Dr. Hwu received his Ph.D. degree in Computer Science from the University of California, Berkeley. His research interests are in the areas of architecture, implementation, and software for high-performance computer systems. He is the director of the OpenIMPACT project, which has delivered new compiler and computer architecture technologies to the computer industry since 1987. He also serves as the Soft Systems Theme leader of the MARCO/DARPA Gigascale Silicon Research Center (GSRC) and on the Executive Committees of both the GSRC and the MARCO/DARPA Center for Circuit and System Solutions (C2S2).

For his contributions to the areas of compiler optimization and computer architecture, he received the 1997 Eta Kappa Nu Holmes MacDonal Outstanding Teaching Award, the 1998 ACM SigArch Maurice Wilkes Award, the 1999 ACM Grace Murray Hopper Award, the 2001 Tau Beta Pi Daniel C. Drucker Eminent Faculty Award, and the 2006 15-Year Most Influential Paper Award of the ACM/IEEE International Symposium on Computer Architecture. He served as the Franklin Woeltge Distinguished Professor of Electrical and Computer Engineering from 2000 to 2004. He is a fellow of IEEE and ACM.

Dr. Hwu received a Phoenix RFP Award in 2005 for his project *Program Visualization with Fulcra and Phoenix* and a Phoenix Direct Funding Award in 2007 for *Moving Future IMPACT Development into the Phoenix Infrastructure*.

Dr. Wen-mei Hwu's paper *Implicitly Parallel Programming Models for Thousand-core Microprocessors* argues that implicitly parallel programming models are critical against software scalability challenges and software development crisis for many-core microprocessors. To read more, this whitepaper is available online at <http://www.crhc.uiuc.edu/IMPACT/ftp/conference/dac44.pdf>.



PROF. DR. ANDREAS POLZE

Operating Systems and Middleware chair
Hasso-Plattner-Institute for Software Engineering
at University Potsdam, Germany
<http://www.dcl.hpi.uni-potsdam.de/>

Email: andreas.polze@hpi.uni-potsdam.de

Andreas Polze is the Operating Systems and Middleware Professor at the Hasso-Plattner-Institute for Software Engineering at University Potsdam, Germany. He received a doctoral degree from Freie University Berlin, Germany, in 1994 and a habilitation degree from Humboldt University Berlin in 2001, both in Computer Science. His habilitation thesis investigates

Predictable Computing in Multicomputer-Systems.

Andreas Polze was a visiting scientist with the Dynamic Systems Unit at Software Engineering Institute, at Carnegie Mellon University, Pittsburgh, USA, where he worked on real-time computing on standard middleware (CORBA) and with the Real-Time Systems Laboratory at University of Illinois, Urbana-Champaign. His current research interests include Interconnecting Middleware and Embedded Systems, Mobility and Adaptive System Configuration, and End-to-End Service Availability for standard middleware platforms.

Andreas Polze received a Phoenix Direct Funding award in 2007 for his research on *Phoenix for Real-time Robotics and Process Control*. You can read more about this project, its research objectives, and the Distributed Control Lab environment below and in accompanying CD-ROM.

PHOENIX TO SUPPORT DOMAIN-SPECIFIC LANGUAGES IN THE DISTRIBUTED CONTROL LAB

The Distributed Control Lab (DCL) provides an open infrastructure for conducting robotics and control experiments from the Web. Users of the DCL can write a control algorithm that can be executed on the physical experiments. Live cameras provide feedback remote users. The lab infrastructure has been successfully integrated into lectures of programming embedded systems including remote lectures at Blekinge Tekniska Högskola in Sweden in Fall 2006. Beside the evaluation of real-time operating systems and programming environments, among them Windows CE.Net and the .NET Compact Framework, the implementation of real-time control algorithms and low-level programming are studied. Within the DCL environment we focus on safety strategies and mechanisms in order to prevent malicious code from damaging experimental equipment. These include source code analysis, domain-specific languages, .NET code access security, runtime observation and the dynamic replacement of faulty control algorithms. Analytical redundancy is used to predictably replace malicious control algorithms by a verified safety controller during runtime. This technique has already been proven in several contexts, the most frequently cited use case is actually the Boeing 777 flight controller. In order to effectively integrate these techniques into our remote lab infrastructure we have implemented tools and a runtime environment supporting dynamic replacement and online monitoring of control algorithms. During the compilation of user control algorithms additional instructions are added with these tools. In order to improve the safe execution of potentially malicious code, we are going to use new compiler tools, including the Phoenix compiler. For the integration of configuration-specific concerns, needed for analytical redundancy, we use Phoenix-based tools, such as our aspect weaver Rapiere-Loom.Net.

For our robotic control experiments we exploit domain specific languages to protect our experiment hardware. Using Phoenix we are able to restrict programming languages. Dangerous construct such as recursion or direct pointer manipulation can be limited. Using these techniques we are able to provide a high availability of our lab experiments.

PHOENIX FOR .NET ASPECT WEAVING

User-defined attributes within the .NET environment allow for tagging language elements with additional information. They have become a widely used and well accepted technology. Throughout .NET, attributes are used to express non-functional properties such as the Serializable attribute which mark a class serializable. Unfortunately the semantics of such attributes is currently described in an informal fashion only. In particular, there is no way to give such attributes additional user-defined behavior. All behavior behind certain attributes is defined within the environmental framework. I.e. the serialization attributes behavior is defined in the .NET framework. Extending the framework to support new attributes with behavior is not readily available yet.

We propose an extension of the idea of user-defined attributes based on the technique of aspect-oriented programming (AOP). AOP was developed to solve the problem of crosscutting concerns. The problem of crosscutting leads to tangled and scattered code. Using AOP it is possible to separate every concern from each other and implement them in discrete units, so-called aspects. Our idea is that attributes with behavior can be seen as aspects: Attributes (aspects) are used to define a non-functional property including the required code. If a language element (which can be an assembly, a module, a class, or a method) is tagged with an aspect, this language element becomes interwoven with this aspect.

Phoenix in our case acts as an intermediate language aspect weaver. Input for the weaver will be a regular .NET-assembly (target assembly) which uses aspects as attributes. The results are two assemblies: a rewritten assembly which references to the interwoven code exactly to these points where aspect attributes were tagged to language elements, and an additional assembly which contains the interwoven code.

PHOENIX FOR EMBEDDED SYSTEMS CODE OPTIMIZATION

The Micro .NET project focuses on executing .NET/ECMA 335 CIL on small embedded systems, which are not covered by existing .NET runtime environments (.NET, Rotor, Compact Framework, SPOT/.NET Micro Framework). These systems typically have limited resources, which prevents the usage of a full-fledged virtual machine, mainly reasoned by the comprehensive class library. Within the Micro .NET project, we investigate execution environments and assembly formats for environments with minimal memory (32kb-256kb). Static linked assemblies are seen as an approach to minimize the memory footprint on the embedded system. The Phoenix framework offers us possibilities to create such optimized assemblies.

In the first step the targeted assembly and the relevant library features are combined into a self-contained binary. The resulting assembly does not require any additional library provided by the CLR. The memory footprint reducing depends on the number of unused features in the former relevant libraries. As a side effect, the process of dynamic linking of features is optimized. The second step evaluates the self-contained assembly to identify unused type features, which may be removed

completely or can be represented without CIL code, while preserving the type layout. The output of this transformation may be an optimized binary assembly representation, like in the .NET Micro Framework. The Phoenix Framework offers us a rich set of functionality to build the Micro .NET tool chain. We are investigating the combination of the .NET language compiler and the Micro .NET tool chain on top the Phoenix technology in our ongoing research.

PHOENIX FOR DYNAMIC WINDOWS RESEARCH KERNEL CODE VISUALIZATION

With the advent of the Windows Research Kernel (WRK), Microsoft released the core part of Windows Server 2003 source code files. This resource will be used in our lab experiments to introduce the Windows operating system to our students.

Understanding the kernel itself is already a challenging task and requests assistance from powerful tools as the Phoenix project. We use Phoenix to develop enhanced analysis tools that instrument certain aspects of the kernel to trace execution flow and to deduce design principles.

In order to trace and analyze kernel behavior during certain operations, we have created a modified, instrumented version of the WRK, called Windows Monitoring Kernel. Phoenix now offers the opportunity to instrument an off-the-shelf retail Windows kernel by modifying the PE file of the kernel. Insights resulting from the WRK can therefore be brought to COTS versions of the Windows kernel. Insights from kernel analysis will be used to develop new operating system abstractions to support service-based systems. For more information on service computing, see the HPI Research School website.



DR. ALFRED AHO

Lawrence Gussman Professor of Computer Science and Vice Chair for Undergraduate Education in Computer Science
Columbia University

Email: aho@cs.columbia.edu

Home page: <http://www1.cs.columbia.edu/~aho/>

Alfred V. Aho is the Lawrence Gussman Professor of Computer Science and Vice Chair for Undergraduate Education in the Computer Science Department at Columbia University. He served as Chair of the department from 1995 to 1997, and in the spring of 2003. Professor Aho has a B.A.Sc in Engineering Physics from the University of Toronto and a Ph.D. in Electrical Engineering/Computer Science from Princeton University.

Professor Aho won the Great Teacher Award for 2003 from the Society of Columbia Graduates. He has won the IEEE John von Neumann Medal and is a Member of the U.S. National Academy of Engineering and the American Academy of Arts and Sciences. He received honorary doctorates from the Universities of Helsinki and Waterloo, and is a Fellow of the American Association for the Advancement of Science, ACM, Bell Labs, and IEEE.

Professor Aho is well known for his many books on algorithms and data structures, programming languages, compilers, and the foundations of computer science. His coauthors include John Hopcroft, Brian Kernighan, Monica Lam, Ravi Sethi, Jeff Ullman, and Peter Weinberger. Professor Aho is the "A" in AWK, a widely used pattern-matching language. "W" is Peter Weinberger and "K" is Brian Kernighan. (Think of AWK as the initial pure version of perl.) He also wrote the initial versions of the string pattern-matching programs egrep and fgrep that first appeared on UNIX.

Professor Aho's current research interests include programming languages, compilers, algorithms, and quantum computing. Professor Aho was Chair of ACM's Special Interest Group on Algorithms and Computability Theory, and is currently Chair of the Advisory Committee for the National Science Foundation Computer and Information Science and Engineering Directorate.

Prior to his current position at Columbia, Professor Aho was Vice President of the Computing Sciences Research Center at Bell Labs, the lab that invented UNIX, C and C++. Previously, he was a member of technical staff, department head, and director of this center. Professor Aho was also the General Manager of the Information Sciences and Technologies Research Laboratory at Bellcore (now Telcordia).

Professor Aho plays bridge, golf, and the violin in a string quartet. He likes skiing on soft snow.

This year, professor Aho received a Phoenix Direct Funding Award for his project *Phoenix Meets Dragon*. Read more about professor Aho's research below and in accompanying CD-ROM.

DEBUGGING ASPECT-ENABLED PROGRAMS

Marc Eaddy¹, Alfred Aho¹, Weiping Hu², Paddy McDonald², Julian Burger²

¹ Department of Computer Science Columbia University New York, NY 10027
{eaddy, aho}@cs.columbia.edu

² Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
{weipingh, paddymcd, julianbu}@microsoft.com

EXTENDED ABSTRACT

The ability to debug programs composed using aspect-oriented programming (AOP) [1] techniques is critical to the adoption of AOP [2, 3]. Nevertheless, many AOP systems lack adequate support for debugging, making it difficult to diagnose faults and understand the program’s composition and control flow. We present an AOP debug model that characterizes AOP-specific program composition techniques and AOP-specific program behaviors, and relates them to the AOP-specific faults they induce. We specify debugging criteria that we feel all AOP systems should support and compare how several AOP systems measure up to this ideal.

We explain why AOP composition techniques, particularly dynamic and binary weaving, hinder source-level debugging, and how results from related research on debugging optimized code help solve the problem. We also present Wicca, which was built using Phoenix, Microsoft’s industrial-strength backend compiler. Wicca is the first dynamic AOP system to support full source-level debugging. We demonstrate how Wicca’s powerful interactive debugging features allow a programmer to quickly diagnose faults in the base program behavior or AOP-specific behavior.

The contributions of this paper are as follows:

- We argue that debugging aspect-enabled programs is more difficult and possibly more important, than debugging conventionally composed programs.
- We present a general model for discussing debugging aspect-enabled programs. The model includes a classification of AOP-specific composition techniques and AOP-specific program behaviors, and a fault model. We define the properties of an ideal AOP debugging solution, including support for *debug obliviousness* and *debug intimacy*.
- We evaluate several current AOP systems as to how well they support AOP de-bugging.
- Since many AOP systems employ source or binary code transformations, we consider how this affects *source-level debugging*, and present solutions suggested by related research on debugging optimized code.
- We present Wicca, our dynamic AOP system that employs a novel weaving strategy to provide full source-level debugging, and is the first dynamic AOP system to do so. We present the results of a debugging experiment using Wicca that demonstrates its unique AOP debugging capabilities.

More information

For the complete whitepaper, see accompanying CD-ROM or visit <http://www.cs.columbia.edu/~eaddy/wicca>.

References

- [1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. TR SPL97-008 P9710042, Xerox PARC, Feb. 1997.
- [2] W. G. Griswold, J. Yuan, and Y. Kato. Exploiting the Map Metaphor in a Tool for Software Evolution. In Proc. of the Intl. Conf. on Software Eng. (ICSE_01), May 2001.
- [3] S. Redwine and W. Riddle. Software technology maturation. In Proc. of Software Eng. (SE'85), Aug. 1985.

WICCA

Marc Eaddy (eaddy@cs.columbia.edu)
<http://www.columbia.edu/~me133>
Columbia University

EXTENDED ABSTRACT

We present Wicca, a research prototype for experimenting with techniques for modularizing programs. Wicca extends our work on the Phx.Morph project, which was built using Phoenix, Microsoft's industrial-strength backend compiler, to enable static and dynamic aspect-oriented programming and open classes. Wicca is the first to allow annotations to be attached to individual program statements, enabling fine-grained statement-level and instance-level advising. Wicca is the first weaver to leverage the .NET 2.0 Debugging APIs to support dynamic AOP, and to support a novel noninvasive breakpoint weaving approach that allows advice to be invoked in-process and to have access to join point context.

Open classes allows a developer to modify class definitions at post-compile time. For example, a developer may wish to extend a class by adding a new field or method. This provides a powerful composition mechanism that provides a greater separation of concerns than what is available with composition mechanisms provided by traditional object-oriented programming languages. Our open classes solution allows any class in a .NET assembly to be extended. Because our implementation is based on assembly rewriting using the Phoenix backend compiler, we can extend assemblies written in any .NET language without requiring access to source code. This makes our solution particularly suitable for extending assemblies developed by third parties.

Aspect-oriented programming (AOP) [1] allows a developer to instrument methods by injecting code or modifying control flow. The developer specifies how to select code for injection (*pointcuts*) and what code is injected (*advice*). Advice code can be injected before a pointcut (*before advice*) and after a pointcut (*after advice*).

References

- [1] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. V. Lopes, C. Maeda, and A. Mendhekar, "Aspect-oriented programming," ACM Computing Surveys, 28(4es):154, 1996.

More information

For a detailed overview, see the accompanying CD-ROM or visit <http://www.cs.columbia.edu/~eaddy/wicca>.

PHX.MORPH

Marc Eaddy (eaddy@cs.columbia.edu)
<http://www.columbia.edu/~me133>
Columbia University

ABSTRACT

We present Phx.Morph, a static byte-code weaver that enables Open Classes and Aspect-Oriented Programming for .NET. Phx.Morph is built on top of Phoenix, Microsoft's industrial-strength back-end compiler framework. Phx.Morph is powerful enough to weave Phoenix itself (1.3 MLOCs), has the unique capability to generate debug information (PDBs), and is semi-officially supported by Microsoft.

Background and Motivation

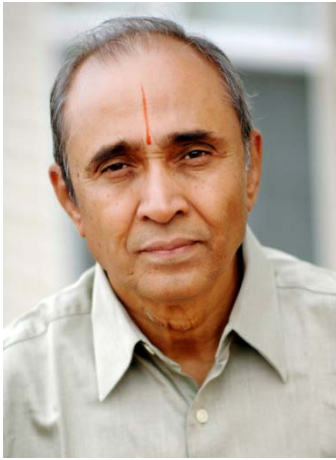
This project started as a research project to explore using open classes as a way for researchers to extend Phoenix compiler classes with their own custom data. For example, a researcher would like to attach their custom analysis data to Instr or Opnd objects created by Phoenix during compilation. Currently, the researcher would need to add their custom data as an extension object that is stored in a list in the Instr or Opnd object. The disadvantages of this approach are: 1) the Instr and Opnd classes require additional code for maintaining and accessing the list, 2) only classes that support this extension list can be extended, and 3) it is inefficient due to the overhead involved in accessing extension objects in the list. In searching for a solution to the problem we decided that open classes was the best approach. In addition to separating concerns nicely the solution is very efficient as field accesses are resolved statically at compile-time. We decided to generalize the approach so that open classes would be available for extending any .NET assembly, not just Phoenix. At some point, we decided to integrate the peweave Phoenix RDK sample into Phx.Morph to support general-purpose AOP.

Goals

Our morphing solution should be *transparent*, *efficient*, and *seamless*. Transparency is achieved by allowing the original developer to be oblivious. They do not need to do anything special to support extensions to their classes. Our solution is efficient because the client is able to statically bind to the newly added fields and methods at compile-time. The solution is seamless for the extension writer because extensions are assemblies themselves and for the client because they are able to use the transformed assembly as they would any other assembly.

More information

For the complete whitepaper, see the accompanying CD-ROM or visit <http://www.cs.columbia.edu/~eaddy/wicca>.



DR. REGETI GOVINDARAJULU

Indian Institute of Information Technology,
Hyderabad, India

Email: gregeti@iiit.net

Home Page: <http://www.iiit.net/~gregeti/>

Dr. Govindarajulu obtained a Ph.D. in Electrical Engineering from IIT Kanpur, 1980. He has been teaching courses in the area of Computer Science and Engineering at undergraduate and postgraduate level for the last 40 years. Has been Professor of Computer Science at NIT (Formerly Regional Engineering College) Warangal, India. Principal, IBM School of Enterprise wide Computing at IIIT Hyderabad and Curriculum Manager at IBM Global Services India. Has been the Dean (Academics) and presently Head (Outreach, Education) at IIIT Hyderabad.

His areas of interest are Computer Architecture, Programming Languages and Compilers. Presently he is teaching courses in the areas of Multicore Processors and Parallel Programming.

Dr. Govindarajulu received a 2005 Phoenix RFP Award for his project *Phoenix-Based Compiler Course Development* and a Phoenix and SSCLI Award in 2006 for his follow-up project *Phoenix-Based Optimizing Compilers Course Development*. You can read more about Dr. Govindarajulu's projects below.

PHOENIX-BASED COMPILER COURSE

I.I.I.T Hyderabad, India.

A study of the Microsoft Phoenix framework was made and the Intermediate Representation (IR) analyzed. A course on compilers taught to undergraduate students at IIIT, Hyderabad focuses on the compiler backend and IR transformations. Microsoft Phoenix environment's modular structure facilitates teaching compiler (course) laboratory courses. The Phoenix framework allows students to access different modules and different phases of the Compiler, making compiler instruction more effective.

Phoenix was used as a platform for compiler course lab assignments at IIIT, Hyderabad during the academic year 2006/2007. Students took up projects on the following techniques:

1. Tail Recursion Elimination.
2. Code Straightening.
3. Loop Invariant Code Motion.
4. Loop Inversion.
5. Unreachable Code Detection and Elimination.
6. If Simplification.
7. Loop Simplification, Loop Unrolling and Value Numbering.
8. Constant Folding and Constant Propagation.

To spread the idea of Phoenix platform for teaching Compilers, a 5 day intensive course on 'Phoenix and IR Transformations' was conducted during December 4-8, 2006 at IIIT, Hyderabad for the teachers from Engineering colleges(in Andhra Pradesh) handling courses on Compilers.

The objective of the course was to expose the faculty on how the Phoenix framework can be used in the laboratory while teaching compilers. Jim Hogg, from Microsoft Research, participated by delivering classroom lectures and conducting lab classes. Course feedback showed that participants appreciated the course and found it interesting.

The compiler course will be offered during the 2007/2008 academic year where established techniques will be analyzed and refined and students will be encouraged to try out more involved optimization techniques. In view of the developments in MultiCore Processors, GPGPUs and Parallel Programming, the compilers will play a key role in software development and performance enhancement.



PROFESSOR VLADIMIR SAFONOV

Professor, Head of Laboratory
St. Petersburg University (<http://www.spbu.ru/e/>)
St. Petersburg, Russia

E-mail: v_o_safonov@mail.ru
Home page: <http://user.rol.ru/~vsafonov>

Professor Safonov is professor of the chair of computer science and head of the laboratory of Java technology in the Department of Mathematics and Mechanics at St. Petersburg University. St. Petersburg University is world-known for its strong scientific schools, particularly in computer science, mathematics and physics. In 2000 and 2001, St. Petersburg students were named world champions in the ACM programming contest.

Professor Vladimir O. Safonov is a specialist in computer science and software engineering. He has 30 years of experience managing and leading major software projects and teaching software technologies at St. Petersburg University. Prof. Safonov has extensive experience in developing commercial software projects in the areas of compilers, expert systems, and Java technology. He has 15 years managing and leading international projects in software development, including managing a team of up to 75 software engineers in St. Petersburg working with Sun Microsystems in the area of compilers and Java. Professor Safonov's most well-known projects are Aspect.NET (<http://www.academicresourcecenter.net/curriculum/Default.aspx?id=6801>), an aspect-oriented programming toolkit for Microsoft.NET based on Phoenix and integrated with Visual Studio.NET 2005, used in about 20 countries, and Knowledge.Net (www.knowledge-net.ru), a knowledge management environment for Microsoft.Net based on C# extensions by ontologies, frames and rule sets.

Since 2002, Vladimir Safonov has been awarded five research and educational grants by Microsoft Research, a teaching grant from Sun Microsystems, and two research grants from the St. Petersburg government. In 2006, professor Safonov received a Microsoft Phoenix and SSCLI award for his project "SPBU for Phoenix – a set of aspect-oriented programming and compiler development tools based on Phoenix". SPBU for Phoenix includes:

- Enhancement of Aspect.NET.
- HL-AST – a high level, language-agnostic, abstract syntax trees architecture targeted to Microsoft Phoenix.
- Phoenix-FETE – a compiler front-end development tool and environment targeted to Microsoft Phoenix.

For more information on this project, see highlights below and accompanying CD-ROM.

HIGH-LEVEL LANGUAGE ABSTRACT SYNTAX TREE FOR PHOENIX

(Phoenix HL-AST)

Vladimir O. Safonov
Professor, Head of laboratory
St. Petersburg University
September 2006

The purpose of the HL-AST project is to develop a more enhanced version of the Phoenix AST. The current Phoenix AST is tied to Microsoft Visual C++ and does not adequately support the needs of other languages. It also should be enhanced by semantic attributes, to fully represent the results of a compiler front-end's activity. In addition, the concept of AST visitor (as now implemented in Phoenix, and previously known in CCI) does not have an efficient implementation. As a result, for languages like C#, VB.NET etc., semantic analysis algorithms may appear to have even exponential complexity. It can dramatically decrease the performance of the compiler. Our idea is to design, first, a language-independent version of AST for Phoenix that will not be tied to C++, second, to dramatically improve the performance of Phoenix-based compilers by developing a version of AST based on our own compiler development experience, especially in efficient semantic analysis techniques and efficient semantic attributes representation and evaluation (patented by three U.S. patents). As part of our HL-AST effort, we plan to develop an editor to visualize and edit HL-AST, and a HL-AST reader which should generate Phoenix High-Level IR (HIR).

Basic principles: change viewpoint on AST as a higher level construct

Abstract Syntax Trees (AST) play a very important part in compilers as a kind of intermediate representation of a program.

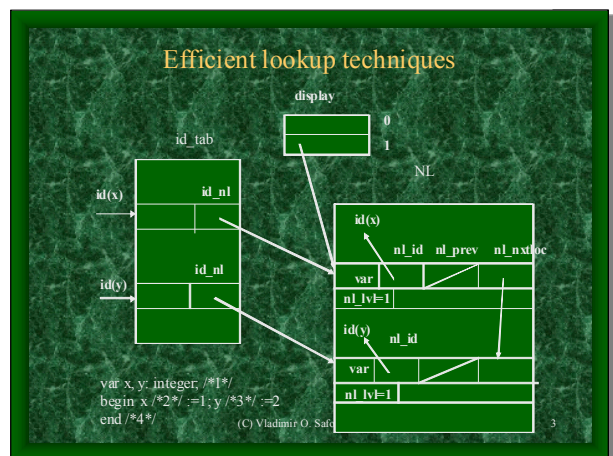
However, in most classical compiler papers, e.g., AST is considered to be just a simplified mapping of the language syntax, with all semicolons, parentheses, etc. omitted.

Our viewpoint on AST, based on our 30-years of experience and results in compiler development is quite different.

Here are the main principles how we think AST should be considered.

1. AST should be attributed, i.e., it should carry semantic attributes (most importantly, the "a-priori" type of each construct). It will help to make the front-end much more efficient, to avoid "exponential" complexity of semantic analysis.
2. AST should be related to efficiently organized front-end semantic tables
3. AST should be language-independent, or, at least, cover a wide enough class of languages – primarily, object-oriented and procedural languages.
4. AST architecture should enable very efficient implementation of visitors (walkers).

Figure 5: Efficient lookup techniques



The principles and collection of ideas presented here are the first step in the creation of a design document. For a fuller explanation and justifications of these principles, please see the accompanying CD-ROM. Any questions, comments or suggestions are greatly appreciated.

PHOENIX FRONT-END TOOLKIT AND ENVIRONMENT

(Phoenix-FETE)

Vladimir O. Safonov
Professor, Head of laboratory

Daniel A. Vassilyev
PhD Student

St. Petersburg University

October 2006

Phoenix.FETE is a modern tool for compiler development based on Microsoft Phoenix being developed at St. Petersburg University under supervision of professor Vladimir Safonov, as part of Microsoft Research granted "SPBU for Phoenix" project.

The main features of the tool are:

- support of combining several parsing and error recovery techniques;
- built-in mechanisms of intermediate code generation (HL-AST or Phoenix HIR).

The purpose of the Phoenix-FETE project is to develop Phoenix-oriented compiler front-end tool and environment based on syntax-directed translation schemes, similar to *yacc*, *bison* and other compiler generation tools, but intended to use with Phoenix. It should take as input an attributed LL or operator precedence grammar of a programming language and produce a compiler front-end for this language that issues HL-AST or Phoenix IR. The purpose of the tool is to make easier a new compiler front-end development for Phoenix.

Features to be implemented in Phoenix-FETE:

- The ability to process either LL or operator precedence grammars and to generate a purely recursive descent, or combined recursive descent (for statements) and operator precedence (for expressions) parser, since in real compilers, including our experience, often combined forms of parsers are used;
- The ability to define and process LL or operator precedence conflicts (like in other modern compiler development tools – ANTLR, Coco/R);
- Support of efficient parsing and semantic evaluation techniques (storing synthesized attributes in the HL-AST during LR parsing, etc.);
- Support of visualization, understanding and control over the process of generation a Phoenix-FETE based compiler front-end, and the process of the generated front-end's work;
- Adequate support of both HL-AST (whose architecture should be developed as another part of our project) generation, and directly Phoenix HIR generation by a Phoenix-FETE generated compiler front-end;
- The ability for Phoenix-FETE to work as a Visual Studio.NET add-in.

For the complete whitepaper, see accompanying CD-ROM.



DR. ERIC WOHLSTADTER

Assistant Professor
The University of British Columbia (<http://www.ubc.ca/>)

Email: wohlstad@cs.ubc.ca
Home page: <http://www.cs.ubc.ca/~wohlstad/>

Eric Wohlstadter is an assistant professor in the department of computer science at UBC. His research interests include: aspect-oriented programming, development and description of web services, and tools for program understanding and maintenance. Eric was the organizing chair of the Aspect-Oriented Software Development (AOSD) 2007 Conference and is a member of the AOSD 2008 programme committee.



DR. KRIS DE VOLDER

Assistant Professor
The University of British Columbia (<http://www.ubc.ca/>)

Kris De Volder is an assistant professor in the department of computer science at UBC. He is interested in software modularity, aspect-oriented programming and in practical applications of logic and declarative languages in the construction of software development tools.

Eric Wohlstadter received a 2005 Phoenix RFP Award for his project *Domain-specific Language for Efficient Design-rule Checking*. To read more about Dr. Wohlstadter's and Dr. De Volder's project, the whitepaper *A Static Aspect Language for Checking Design Rules* is available online at <http://portal.acm.org/citation.cfm?id=1218571>.



DR. CHEN DING

Associate Professor
University of Rochester (<http://www.rochester.edu/>)

Email: cding@cs.rochester.edu

Home page: <http://www.cs.rochester.edu/~cding/>

Chen Ding is an Associate Professor in the Department of Computer Science at the University of Rochester, Rochester, New York. He received his Ph.D. from Rice University, M.S. from Michigan Tech., and B.S. from Beijing University. For research he is interested in modeling the composite and emergent patterns in large scale program behavior, including program locality, reference affinity, and program phases. Based on these models, he develops software techniques for program transformation, memory management, dynamic parallelization, and behavior-oriented programming. He received the Early Career Principal Investigator award from the Office of Science of the U.S. Department of Energy, the CAREER award from National Science Foundation, the Faculty Fellowship of IBM Center for Advanced Studies, and a best-paper award for his work with Ken Kennedy on compiler enhancement of global cache reuse. With Trishul Chilimbi and Frank Mueller, he co-organized the first ACM SIGPLAN Workshop on Memory System Performance (MSP). He is a visiting researcher at Microsoft Research Redmond from February to August 2007.

Dr. Ding received a 2006 Phoenix and SCLI Award for his project *Adaptive Heap Size Control Using Phoenix and .Net Virtual Machine*. Read more about Dr. Ding's work below and in accompanying CD-ROM.

WASTE NOT,WANT NOT: ADAPTIVE GARBAGE COLLECTION IN A SHARED ENVIRONMENT

Chengliang Zhang[†], Kirk Kelsey[‡], Xipeng Shen[#]
Chen Ding[†], Matthew Hertz^{_}, and Mitsu Ogihara[†]
[†]Department of Computer Science
University of Rochester
{zhangchl,kelsey,cding,ogihara}@cs.rochester.edu

[#]Department of Computer Science
The College of William and Mary
xshen@cs.wm.edu

^{_}Department of Computer Science
Canisius College
hertzm@canisius.edu

University of Rochester Computer Science Department Technical Report TR-2006-908

Introduction

Software developers are taking advantage of garbage collection for the many engineering benefits it provides using either garbage-collected languages such as Lisp, ML, Java, and C# or conservative garbage collectors such as the Boehm-Demers-Weiser collector [6]. While a conventional program uses exactly as much memory as it needs, the memory use of a garbage collected program can expand

beyond its need. This difference raises the possibility of *resource-based garbage collection*, which is to adapt the frequency of GC in response to the changing amount of available memory.

Memory usage depends first on the program demand. The heap must be large enough to accommodate all reachable data. When the heap size is large enough, setting a larger heap size in the virtual machine will lead to fewer calls to the collector, which means lower collection time, so long as the heap size does not exceed the available memory and cause paging. The common scheme for Java programs is to use a fixed range by which the heap size can exceed the program's need. As an example Jikes RVM uses a range of 50MB to 100MB by default. This is often a mismatch to the available resource for a number of reasons. First, the actual memory usage depends on the garbage collector implementation (e.g. twice the heap size for a copying collector), not counting the memory needed by the virtual machine and the operating system. Second, knowing the exact amount of available memory is difficult because it requires ascertaining the active memory usage of other programs. Much memory may be occupied but not used (for example the file cache) and can be made available. Last but not least, the heap size, which is set before the execution, cannot respond to change of conditions in the middle of an execution.

In this paper we address the problem of resource-based garbage collection in a shared environment, which is increasingly common on today's multi-processor, multi-core, and multi-threaded machines. Garbage-collected programs, however, lack a firm basis for high degrees of multi-programming if two problems remain unsolved. The amount of available memory may change significantly and dynamically because of memory sharing, making any setup vulnerable to extreme and adverse conditions if it does not adapt to the dynamic resource. Also, the moment a program starts to adapt, it embarks on a collision course with other like-minded programs, no matter how much memory is available.

We first revise the classical performance model, which is demand-based, to accommodate resource-based memory management. We introduce a new quantitative notion called *time-memory curve* and use it to predict the performance of memory sharing in this new context. An example of a resource-based garbage collection system is *program level adaptive memory management (PAMM)*, which uses run-time monitoring and heap-size control for resource adaptation [24]. We develop a scheme similar to PAMM for memory sharing. While the previous scheme used semi-manual phase analysis, we make it fully automatic so it can be applied to general programs. The previous scheme is designed for a static environment and by nature is *selfish*. It does not consider other contenders for the monopoly on memory. We present *cooperative PAMM*, which dynamically divides the available memory among the resource contenders through a shared white-board data structure to maximize system throughput.

The adaptive GC techniques are implemented at the program level by inserting a run-time monitor and controller, without users' effort. They monitor the program demand and the memory performance (in particular the memory paging) to adapt the memory demand with the available resource. They do not require changes to the virtual machine or the operating system, so they can be automatically deployed on existing systems.

More information

For the complete whitepaper, see accompanying CD-ROM

References

[6] H.-J. Boehm and M. Weiser. Garbage collection in an uncooperative environment. *Software: Practice and Experience*,

[24] C. Zhang, K. Kelsey, X. Shen, C. Ding, M. Hertz, and M. Ogihara. Program-level adaptive memory management. In

Proceedings of the International Symposium on Memory Management, Ottawa, Canada, June 2006.

MICROSOFT PHOENIX RESEARCH AWARDS

The Microsoft Phoenix Academic Program has supported over forty research projects since 2003. The list below describes each of these projects and their principal investigator.

PHOENIX EARLY ADOPTER AWARDS – 2004

Binary Optimization with Phoenix based on Phase Behavior

Brad Calder

University of California - San Diego

Research involving Phoenix as a framework for optimization and analysis of binaries based on phase behavior.

Phoenix Text

John Gough

Queensland University of Technology

Research and teaching material documenting the accessibility of the Phoenix framework and its plug-ability. As well as the ability to build program instrumentation tools and tools for code checking.

Phoenix Optimization Infrastructure

Michael Smith

Harvard University

Research involving the Stanford University Intermediate Format compiler system optimization infrastructure ported to Phoenix.

Phoenix Program Slicing

Rajiv Gupta

University of Arizona

Research involving the use of Phoenix for optimization using program slicing.

PHOENIX RFP AWARDS – 2005



Constructing Compact Debugging Traces with Binary Code Analysis and Instrumentation
Yinong Chen
Arizona State University

We will use Phoenix to apply novel slicing techniques to automatically generate compact effect-cause traces, which have wide applications to debugging, profiling, and monitoring.



Phoenix-Based Compiler Course Development
Regeti Govindarajulu
Indian Institute of Information Technology, Hyderabad

The work involves enhancing an undergraduate compiler curriculum to include more sophisticated backend and optimization content using Phoenix as the backend framework.

Compiler Backend Experimentation and Extensibility Using Phoenix
Suresh Jagannathan
Purdue University

This project aims to leverage tools and technologies available in the Phoenix and .NET framework to study compiler extensibility and analyses. First by using MLton, an optimizing compiler for StandardML. We will target MLton's first-order IL to Phoenix and leverage existing tools to more easily support retargetability, profiling, and improved backend optimizations. Conversely, we will port many of MLton's current optimizations to the Phoenix environment, to allow these optimizations to be applied elsewhere as appropriate. In particular, we will integrate compiler optimizations available in MLton to F#, and will leverage F# utilities and interoperability features to improve MLton, by retargeting the F# front-end to generate a MLton-readable IR. Second, we propose to implement necessary compiler support using Phoenix. By using the Phoenix and .NET framework, we believe a lightweight transactional model for C# is feasible. Such an implementation would be especially valuable in scalable multiprocessor environments. We would also explore opportunities to investigate the interaction of lightweight transactional models with C# extensions such as Polyphonic C# that provide asynchronous methods and synchronization patterns.



Adaptive Inline Substitution in Phoenix
Keith Cooper
Rice University

Inline substitution is a simple and powerful code transformation that has the potential to improve program performance in a variety of circumstances. The mechanism of inline substitution is straightforward. The compiler replaces a procedure call (a "call site") with the body of the called procedure (the "callee"). The resulting code often has properties that lead the compiler to produce more efficient code than it would for the original code that used separate procedures and a call.

We have an ongoing investigation into the use of adaptive techniques to choose specific call sites to inline. We propose to move this work into Phoenix, where it can capitalize on the stable infrastructure of the Phoenix system. A critical part of this work involves experimenting with different static and dynamic measures of the program. Phoenix's analysis and profiling capabilities will let us perform a broader set of experiments with less implementation overhead. Finally, by using Phoenix, we can scale our experiments to large-scale systems—a critical part of demonstrating success with inlining since many of the interesting effects of inline substitution only arise in large programs.



Domain-Specific Language for Efficient Design-Rule Checking
Eric Wohlstadter
The University of British Columbia

This project addresses the problem of checking source code against a set of coding conventions, commonly referred to as *design rules*. Enforcement of design rules is becoming an important practice in industry as evidenced by the release of Microsoft's "Design Guidelines for .NET Class Library Developers". Design rules are not (and should not) be directly part of a programming language semantics. This is because design rules can be highly project or task specific. For this reason Microsoft has released a tool, FxCop, for design-rule checking. FxCop is extensible through an API that exposes an intermediate representation of code to design-rule developers. However, programming design rules can be difficult and error prone even when a programmer is familiar with the details of the intermediate representation. This project addresses this problem by building a domain-specific language tailored for the purpose of specifying design rules. We intend to make use the domain-specific language analysis to optimize sets of design-rule checks over code. Our approach is based on our prior work with the JQuery program database. Develop a domain-specific language to allow developers to express "Design Rules for Modularity." The language allows the expression of patterns that generally constitute symptoms of bad modularity "code smells" and scoping rules that describe the desired modular structure of a software system.

Setpoint: An Aspect Oriented Framework Based on Semantic Pointcuts
Victor Braberman
Universidad de Buenos Aires

We propose an innovative approach to attain the two basic constituents of AOSD: quantification and obliviousness. This approach consists in annotating source code with semantic information through metadata, which can later be used in the construction of semantically rich pointcuts to guide aspect weaving: setpoints. Traditional AOP frameworks impose rigid syntactic conventions to the source code or require ad-hoc artifacts to adapt generic aspects to particular applications. We firmly believe that replacing these restrictions and adapters with explicit semantics will remove the main obstacle that keeps AOP technologies from becoming a massively adopted resource for systems development. Phoenix will allow us to integrate the so-called preweaving and semantification processes into the environment, maintaining SetPoint implementation isolated from IL management internals. This synergy will be achieved in a natural way, inserting these processes as phases in program generation. Phoenix will help us explore another semantic AOP need as well: semantic contracts.



Phase Aware Profiling with Phoenix
Chandra Krintz
University of California at Santa Barbara

The goal of our research is to use the Microsoft Phoenix Framework to enable transparent, software-based, post-deployment, program optimization, bug isolation, and coverage testing. We have shown in prior work that programs commonly do not behave randomly but instead, execute as a series of phases. During a particular phase, the behavior of the program is relatively stable for some amount of time, after which the behavior may drastically switch. Furthermore, these same phases may then re-occur at some point later in time. We propose to exploit this phase behavior within the Phoenix system to improve the efficiency and efficacy of remote profiling. To enable this, we will develop intelligent sampling techniques that sample each phase of a remotely executing program instead of continuously sampling (randomly, periodically, or otherwise) over the lifetime of the program.

Using Call Graph Analyses to Guide Selective Specialization in Phoenix
Cormac Flanagan
University of California at Santa Cruz

This projects plans to extend the Phoenix compiler framework and run-time system to provide support for lightweight abortable transactions. The run-time system will provide a transaction API through which the source program can indicate the dynamic scope of transactions. (This approach avoids the need for syntactic changes in the source language.) If a transaction encounters an error or other unexpected event, it can abort via an appropriate call to the transaction API. In this case, the modified Phoenix run-time system is responsible for rolling-back the program to its consistent, pre-transaction state. Where possible, external events such as file system operations, are also rolled-back.



Program Visualization with Fulcra and Phoenix
Wen-Mei Hwu
University of Illinois at Urbana-Champaign

We propose to create a suite of safe, accurate, scalable and intuitive program understanding tools in Phoenix. Critical to the fulfillment of this goal are safe, high resolution pointer analysis tools—tools conventionally assumed not to scale to the size of typical commercial applications. Having developed Fulcra, an accurate and efficient pointer analysis system, and demonstrated its safety, accuracy and scalability on SPEC and other benchmark applications, we are now ready to address a larger scope of programs. This project will interface Fulcra with Phoenix to enable accurate analysis and visualization of large-scale production software. By performing online summary compaction and local adaptation in a flexible constraint framework, Fulcra is the first pointer analysis system to provide context sensitivity, truly safe field sensitivity, heap object cloning, and inclusion (subtyping) together in a scalable implementation. Among the SPEC CPU benchmarks, we have shown the system to provide much higher accuracy than traditional unification-based approaches while maintaining a developer-friendly run time. Phoenix integration will allow the evaluation of Fulcra’s accuracy and scalability on Microsoft’s large, production software and will facilitate rapid development of a practical application of Fulcra in a source-level visualization tool. We expect this project will ultimately demonstrate the practical applicability of “deep” static analysis in the construction of useful code understanding tools.

Navel: Automating Software Support Using Traces of Software Behavior

Emmett Witchel
University of Texas at Austin

Use Phoenix as the instrumentation engine for large, real-world client or server applications to insert probes that identify program behavior and provide a fingerprint to classify and identify software failures. Software robustness and software support are crucial problems today and the state of the art is disappointing. Navel is a project set to improve life for end users and companies that provide software support. When software fails, end users often have no recourse but to contact a call center. Call center employees often rely on nothing more than responses to a small set of pre-planned questions. This form of support works only for a limited class of failures, and results in great frustration when problems are not resolved. Recently, crash diagnosis systems that gather information at the end user's machine and send it to the vendor are an improved direction for software support. However, these systems are not yet making sophisticated choices about what data to send. As a system, Navel will broaden the scope of system behavior which generates feedback, it will enrich the information collected and sent, and it will include algorithms to interpret that information. The information that Navel collects will enable call center employees to make more accurate diagnoses, or it might automate diagnosis for some issues. Information tracked by Navel could also be fed back to program developers who can focus bug fixing on problems that occur in the field. Navel will insure that more useful information is sent when software malfunctions, and that more can be done with that information by support staff and developers.



Techniques and Tools for Software Assurance

Jack Davidson
University of Virginia, USA

The primary goal of our research is to develop robust, affordable, scalable techniques for providing higher levels of software assurance. The expected outcomes, using the Phoenix infrastructure, include: a set of synthetic test cases for gauging the effectiveness of techniques at countering malicious code attacks, experiments to understand how and what types of malicious code can be inserted into Windows binaries, robust binary security transformations for protecting Windows binaries against attacks, a security testing framework and tool that enables testing for vulnerabilities before deployment, and experimental evaluations that demonstrate the effectiveness and efficiency of the testing framework.

Type-Checking the Intermediate Languages in the Phoenix JIT Compiler

Zhong Shao
Yale University

Our project is to design and implement a sound type system for the intermediate representation of Phoenix. A sound type system will allow a way to automatically check that the result of compilation will not crash unexpectedly.

PHOENIX AND SSCLI AWARDS – 2006

Compiler Support for Software Transactional Memory Maurice Herlihy Brown University, U.S.

We propose to develop a Phoenix-based compiler plug-in that will transparently transform sequential objects into properly synchronized concurrent ones. There are two broad areas where a compiler extension for SXM would make the API substantially more attractive. First, it could provide a safer and easier-to-use interface to the programmer. Second, it would permit pervasive optimizations that are simply not possible using a library-based implementation.



Phoenix-Based Optimizing Compilers Course Development Regeti Govindarajulu Indian Institute of Information Technology, Hyderabad, India

The project is aimed at developing course-material for teaching a course on optimizing compiler with a strong laboratory component. The following are the expected outcomes: Development of course material for optimizing compilers with strong practical assignments, Development of phoenix-based lab exercises in the form of plug-ins and tools, The laboratory modules developed will be made freely available to other institutes.

Integrating Dynamic Slicing into the coredbg Debugger Neelam Gupta University of Arizona, U.S.

We aim to enhance the existing coredbg debugger to perform reverse execution, i.e. to step backwards during the course of a program's execution. We will use this basic facility to implement demand-driven implementations of dynamic slicing algorithms. Variants of dynamic slices (data slices, full slices, and relevant slices) will be supported. Once we have implemented these algorithms, we will then implement additional dynamic analyses that will enable selection and pruning of dynamic slices.

A Testbed for Studying the Order and Combination of Code Optimization Phases Michael Smith Harvard University, U.S.

The ultimate goal of our research is to create an experimental testbed that directly helps us to increase our fundamental knowledge of code optimization, improve our understanding of the interaction between optimizations, and identify profitable directions for new combined optimizations.

PTV: Translation Validation in the Phoenix Compiler Framework

Lenore Zuck

University of Illinois at Chicago, U.S.

Benjamin Goldberg

New York University

There is a growing awareness, both in industry and academia, of the crucial role of formally proving the correctness of systems. Most verification methods focus on the verification of a specification with respect to a set of requirements, or of high-level code with respect to a specification. Our ultimate goal is to develop a methodology for the translation validation of advanced optimizing compilers built upon frameworks, like Phoenix, that provide a multi-language and multi-platform (e.g. managed and unmanaged code) environment.



Phase Detection and Optimization

Chandra Krintz

University of California at Santa Barbara, U.S.

As part of the project that we propose herein, we plan to build upon and extend our prior work to investigate two primary research foci: Online detection of phase behavior and phase-aware feedback-directed optimization. We plan to investigate and implement our techniques in Phoenix (for phase capture as well as phase-aware optimization for a static compiler via Phoenix plugins) and SSCLI (for phase-aware dynamic compilation and optimization).

Extending Dynamic Features of the SSCLI

Francisco Ortin

University of Oviedo, Spain

This research topic is based on the need to dynamically modify running applications in systems that must be adapted to changes in their environment without being halted, modified, recompiled and restarted. Examples are long-running systems or monitoring and debugging components. AOP is based on program transformation and in DAOP this transformation takes place at runtime. The capabilities needed to create DAOP applications are offered by dynamic languages by means of structural reflection, dynamic code generation and dynamic typing features.

A Lua Compiler for the Phoenix Framework

Fábio Mascarenhas

Pontifícia Universidade Católica do Rio de Janeiro, Brazil

We expect to produce a fully working compiler for the Lua language, fully implemented in managed code, with both CIL and Phoenix IR back-ends.



Developing a Testing Framework for Security
Mary Lou Soffa
University of Virginia, U.S.

The overall goal of this research is to develop a testing framework for detecting and managing security flaws. The key idea is to develop a set of static analyses that are path sensitive and demand driven and can be used to determine programs paths that lead to various types of vulnerabilities.



Using Phoenix in Anti-Virus Curricula
Jack Davidson
University of Virginia, USA

An anti-virus software course that uses Phoenix in the programming assignments can introduce students to the use of compiler program analyses in the detection of malicious code.

A Viable Approach to Compiling Sequential Codes for CMPs
David August
Princeton University, U.S.

Instruction-level data flow driven program refactoring is a promising approach to extracting decoupled pipeline parallelism from sequential programs. A fairly straight-forward implementation is able to extract parallelism from many SPEC benchmarks and generate speedups of up to 20% on whole programs. In this work, we expect to fully develop compiler technology that will produce speedups many times the original proof of concept.

Improving the Compilation of Lazy Functional Languages Using Phoenix and the SSCLI
Andre Santos
Federal University of Pernambuco, Brazil

We propose to continue the development of the Haskell compiler for .NET, pursuing a detailed analysis on the performance bottlenecks we have found, using the new SSCLI version and Phoenix. For Phoenix, specifically, we intend to integrate it to the backend of the compiler, in order to use it as a code generator for the compiler and also as our code instrumentation and analysis tool.

A Phoenix-Based Tool for Data Flow Testing
Dragan Bojic
University of Belgrade, Serbia and Montenegro

The main outcome of the project should be a data flow testing tool, based on Phoenix RDK. The tool will support dataflow test-adequacy criteria such as: def-use, c-use, p-use, and all-use. We will also consider implementing other criteria as well, such as all du-paths or interprocedural data flow testing (global def-uses).

Concurrency Support for Managed Code and Interactive AsmL

Dean Rosenzweig

University of Zagreb, Croatia

We propose two variants of the project, depending on available resources. The first variant includes SSCLI based runtime support for efficient execution of finely grained potentially concurrent code over a transactional memory and a Phoenix-based AsmL compiler. The second variant extends the first one with optional support for contracts and plug-in architecture for external reasoning tools.



SPBU for Phoenix (SPBU4PHX): A Set of Compiler Development and AOP Tools Based on Phoenix

Vladimir Safonov

St. Petersburg University, Russia

The purpose of the project is to develop a set of state-of-the-art compiler development and aspect-oriented programming tools based on Phoenix, as an enhancement of our previous experience of developing Aspect.NET project on top of Phoenix, and of our 30 years experience in compiler development area.



Adaptive Heap Size Control Using Phoenix and .NET Virtual Machine

Chen Ding

University of Rochester, U.S.

At present, the task of memory management for garbage collected programs rests completely at the control of the virtual machine. The goal of this research is to explore ways where information of program behavior patterns could be used to augment the default management policies. It will lead to program-specific policies that adapt to the need of individual programs.

Region Memory System for Scalable Performance

Wei-Ngan Chin

National University of Singapore

We propose region-based memory system (as a complement to garbage-collected heap) whereby objects with similar lifetimes are placed into the same region, whenever possible. We explore techniques to make region inference more precise and predictable for scalable performance to be achieved for large-scale applications. We intend for this project to be conducted under the Phoenix/SSCLI platform to allow support for a wide range of processor architectures, including those suitable for real-time and embedded systems.

PHOENIX DIRECT FUNDING AWARDS – 2007



Phoenix for Real-Time Robotics and Process Control
Andreas Polze
University of Potsdam

Step-by-step lab instructions supported by video snippets (podcasts) that will involve tools and software frameworks for dynamic system update (analytic redundancy) developed in the DCL space ported to Phoenix technology. The multimedia teaching materials will be made widely available through a curriculum repository site and presented at international conferences.



Moving Future IMPACT Development into the Phoenix Infrastructure
Wen-Mei Hwu
University of Illinois-Urbana-Champaign

This research will complete the migration of IMPACT development to Phoenix. The project will deliver a visualization tool to present reader-writer analysis results for code and data objects to developers in a meaningful form. This tool will demonstrate the safety, accuracy, scalability and descriptiveness of our analysis and the aptness of Phoenix in constructing such high-impact tools. Future IMPACT development in Phoenix will also provide generally useful, highly configurable multi-core parallelization information in the Phoenix framework.



Software Testing for Security Vulnerabilities
Mary Lou Soffa
University of Virginia

The overall goal of our research project is to develop robust, flexible and scalable techniques for determining security flaws before software release. The expected outcomes using the Phoenix infrastructure, the Disolver constraint solver, and the SSCLI and Windows Research Kernel case studies are expected next year.



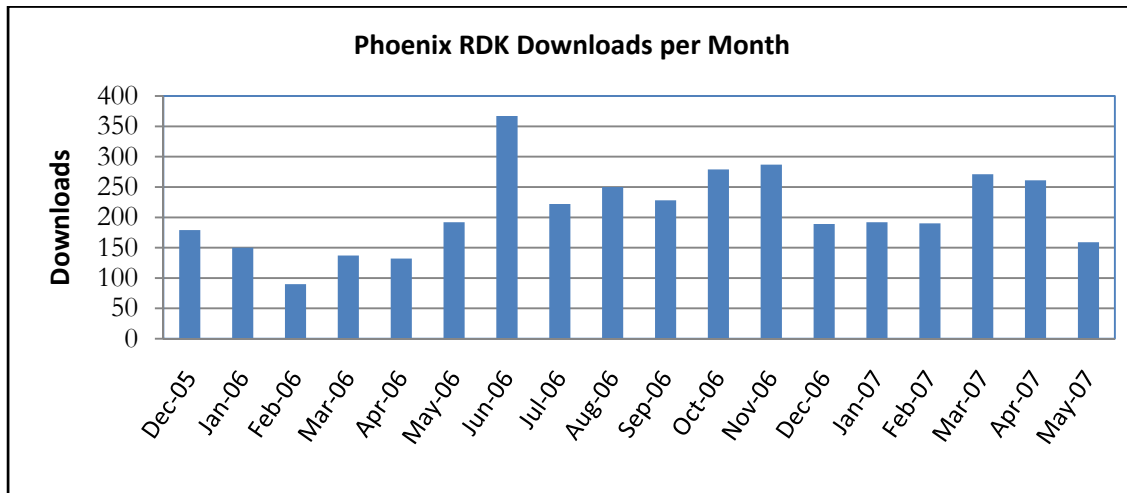
Phoenix Solution of the Dragon Book
Al Aho
Columbia University

The project will use Microsoft Phoenix to create solutions for the exercises in “Compilers: Principals, Techniques, and Tools (2nd Edition)”, or the “The 2007 dragon book”. The goal is to create solutions that are robust and easy to use in the classroom. Potentially, the outcome can be used for teaching advanced compiler courses at universities.

PHOENIX DOWNLOADS

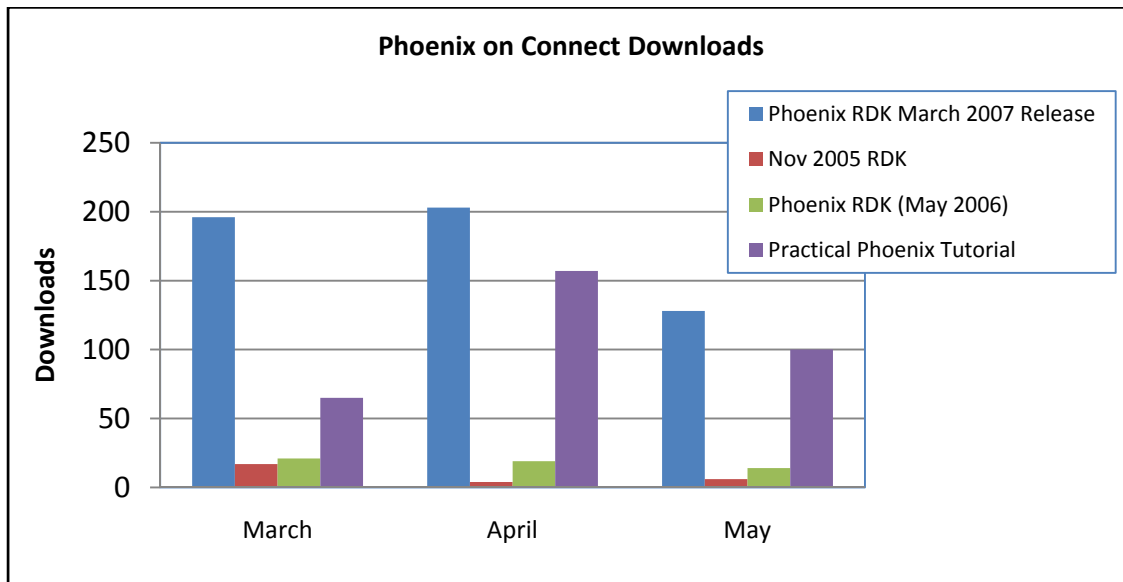
There have been over 4,000 Phoenix RDK and over 300 Phoenix tutorial downloads through the Phoenix Academic program. Figure 6 shows the total RDK downloads per month via the Phoenix Academic Program website (<http://research.microsoft.com/Phoenix>).

Figure 6: Downloads via Phoenix Academic Website



In March 2007, downloads became available through the Phoenix Connect website (<http://connect.microsoft.com/phoenix>). Figure 7 below shows download data through the Phoenix Connect website.

Figure 7: Phoenix Downloads via Phoenix Connect Website



OTHER RESOURCES

Phoenix Research Website

<http://research.microsoft.com/Phoenix>

Phoenix Connect Website

<http://connect.microsoft.com/Phoenix>

Phoenix Downloads

<https://connect.microsoft.com/Phoenix/Downloads>. A free Windows Live ID is required for download.

MSDNAA

<http://msdn2.microsoft.com/en-us/academic/default.aspx>

The Microsoft Curriculum Repository

<http://www.academicresourcecenter.net/curriculum/facetmain.aspx>

SSCLI

<http://research.microsoft.com/sscli/>

PHOENIX USERS FORUM

The Phoenix MSDN Users forum is your primary Phoenix support resource.

<http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=1328&SiteID=1>

Microsoft does not guarantee the accuracy of any information presented and assumes no liability arising from your use of the information. Information in this document, including URL and other Internet Web site references, is subject to change without notice. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

All trademarks are the property of their respective companies.

©2007 Microsoft Corporation. All rights reserved.

Microsoft®
Research

Phoenix Academic Program



PHOENIX
CODEGEN . OPTIMIZATION . ANALYSIS

External Research & Programs
<http://research.microsoft.com/Phoenix>