Academic Press is an imprint of Elsevier

32 Jamestown Road, London, NW1 7BY, UK
Radarweg 29, PO Box 211, 1000 AE Amsterdam, The Netherlands
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
525 B Street, Suite 1900, San Diego, CA 92101-4495, USA

First edition 2010

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-0-12-381027-4

ISSN: 0065-2458

Printed and bound in USA

10 11 12 10 9 8 7 6 5 4 3 2 1

Contributors

Dr. Joshua N. Adkins is a Staff Scientist at Pacific Northwest National Laboratory (PNNL) and his research centers on the comprehensive characterization of proteins through space (associated proteins, structural determinants, and localization) and time (before and after treatment, cell cycle, day–night cycle, and evolutionary changes) to better understand biological systems. Of particular interest to Dr. Adkins are challenging biological studies that require bridging the gaps between technology development and biological application. He leads a talented and multidisciplinary team comprised of scientists from PNNL, universities, and other research organizations whose aim is to develop a systems-level understand of the causative agents of Typhoid Fever and the Black Plague.

Douglas J. Baxter is a Senior Research Scientist at the Department of Energy's Pacific Northwest National Laboratory working as a consultant in the Visualization and User services group for the Molecular Science Computing Facility in the Environment Molecular Sciences Laboratory. He has been developing software for parallel high-performance scientific computing for over 25 years and has currently involved in computational biology (MSPolygraph and Scalablast), some climate modeling, subsurface chemistry, and power grid computations. He received his M.S. in Computer Science from Stanford University.

Dr. William R. Cannon joined PNNL in January 2000, where his current research interests are in (1) simulation of cellular processes such as replication and recombination, (2) statistical and biological inference of cellular networks, and (3) development of a Web-based problem-solving environment for bioinformatics, especially with regard to synchronous processes in the cell.

Dr. Daniel G. Chavarria-Miranda is a Research Scientist at the Department of Energy's Pacific Northwest National Laboratory working in the areas of scalable programming models, multithreaded systems, compilers, and programming languages. He is co-Principal Investigator of the Center for Adaptive Supercomputing Software (CASS-MT), who is conducting research on the use of multithreaded

systems for nontraditional parallel applications, and he is also Principal Investigator for PNNL's portion of the DOE/ASCR Center for Scalable Parallel Programming Models. Dr. Chavarría-Miranda is a member of the Association for Computing Machinery (ACM). He earned his Ph.D. in Computer Science from Rice University.

Sutanay Choudhury is a Computer Scientist in Department of Energy's Pacific Northwest National Laboratory working on High-Performance Computing and Scientific Data Management. His areas of interests include parallel and distributed computing techniques, data mining, and machine learning.

Ian Gorton is the chief architect for Pacific Northwest National Laboratory's Data-Intensive Computing Initiative. His research interests include software architectures and middleware technologies. He received a Ph.D. in Computer Science from Sheffield Hallam University. Gorton is a member of the IEEE Computer Society. Contact him at ian.gorton@pnl.gov.

Deborah K. Gracio is a computational and statistical analytics division director at Pacific Northwest National Laboratory. Her research interests are integrated computational environments and computational biology. She received an M.S. in Electrical Engineering from Washington State University. She is a senior member of the IEEE and the American Association for the Advancement of Science. Contact her at Debbie.gracio@pnl.gov.

Todd D. Halter is a Senior Scientist at the Department of Energy's Pacific Northwest National Laboratory with over 20 years experience in the fields of Computer Science, Physics, Mathematics, and Chemistry. Since 1998, Todd has been working on the Atmospheric Radiation Measurement (ARM) Program—the largest global change research program supported by the US Department of Energy as a value-added product and data system developer and project manager. Todd has served as a group manager, project manager, system architect, and developer with experience in all aspects of staff and project management, budget and resource management, architecture and system design, coding, testing, installation, and maintenance for several other projects within PNNL. Todd holds an M.S. in Computer Science from Washington State University.

Navdeep D. Jaitly received his M.S. degree in Computer Science from the University of Waterloo. He worked as a Software Developer in IBM Toronto Labs, as a Senior Research Scientist and Group Leader in Bioinformatics at Caprion Pharmaceuticals in Montreal and as a Senior Research Scientist at Pacific Northwest National Laboratory. His research interests include machine learning and statistics

and he has spent several years in their application to mass spectrometry. He is currently a graduate student in Computer Science at the University of Toronto where he is pursuing research in machine learning.

John R. Johnson leads high-performance computing (HPC) activities for the Pacific Northwest National Laboratory's Defense and Special Programs. John's current research is in data-intensive, irregular applications requiring novel HPC approaches, for example, streaming sensors, complex networks, and multimedia analysis. Prior to joining PNNL, John worked on portfolio construction and optimization for a large global investment firm. Before that John was a Research Program Manager at Lawrence Livermore National Laboratory leading programs exploring computing architectures and progressive algorithms for data-intensive applications. John leads the CS team for a massively parallel, high-fidelity physics simulation project under the Department of Energy's (DOE) ASCI program where he received a DOE award for technical excellence and a technical achievement award from the National Nuclear Security Administration. John has a B.A. in Mathematics from the University of California, Berkeley, an M.S. in Computer Science from Johns Hopkins University, and Ph.D. studies at the University of Chicago.

Dr. Holger M. Kienle holds a Ph.D. degree from the University of Victoria, Canada (2006), a Diploma in Computer Science from the University of Stuttgart, Germany (1999), and a Master of Science degree in Computer Science from the University of Massachusetts, Dartmouth (1995). He received a 2-year Post Graduate Research Fellowship (1997–1998) from the University of California, Santa Barbara, to work as a member of Professor Hölzle's Object-Oriented Compilers group. He is currently a postdoctoral student in Computer Science at the University of Victoria, Canada, where he is a member of Professor Müller's research group. His interests include software reverse engineering, domain-specific languages, virtual worlds, and legal issues that impact information technology. He is program co-chair for WSE 2010 and co-organizer of the WASDeTT Workshop series. He has served on the program committees of CSMR, ICSM, WCRE, and WSE. His e-mail address is kienle@cs.uvic.ca.

Dr. Richard T. Kouzes is a Laboratory Fellow at the Department of Energy's Pacific Northwest National Laboratory working in the areas of nonproliferation, homeland security, and computational applications. He is an adjunct Professor of Physics at Washington State University, a Fellow of the Institute of Electrical and Electronics Engineers, and a Fellow of the American Association for the Advancement of Science. He earned his Ph.D. in Physics from Princeton University and is the author of over 300 papers. His e-mail is rkouzes@pnl.gov.

Prof. Alfred W. Loo is an Associate Professor at the Department of Computing and Decision Sciences, Lingnan University, Hong Kong. He is also a Chartered Engineer, Chartered Mathematician, Chartered IT Professional, and Chartered Scientist. His research interests are in the areas of wireless security, peer-to-peer systems, and distributed computing. He can be reached at alfred@LN.ln.edu.hk.

Matthew C. Macduff is a Computer Engineer at the Department of Energy's Pacific Northwest National Laboratory with a focus on network, system, and software architecture. He has worked extensively for 16 years with data management systems as an administrator, programmer, designer, and task leader. He is most familiar with Linux and Solaris systems and Cisco network equipment. His current interest is the Scientific Data Management capabilities as they apply to the DOE Atmospheric Radiation Measurement (ARM) project.

Prof. Ami Marowka is an adjunct Assistant Professor in the Department of Computer Science of Bar-Ilan University, Israel. Before joining to Bar-Ilan University he was an Assistant Professor in the Department of Software Engineering of Shenkar College of Engineering and Design. Before joining Shenkar he was an Associate Researcher in the Department of Computer Science of University of Houston and a member in the High-Performance Computing Tools Group (HPCT). Professor Marowka earned his Ph.D. degree in the School of Computer Science and Engineering at the Hebrew University of Jerusalem, Israel. His primary research interest is in portability of high-performance applications. Other research areas include parallel computing, use of advanced computer architectures, programming methodology, tools for parallel computers, and *ad hoc* networks. His e-mail address is amimar2@yahoo.com.

Dr. Andres Marquez is a Scientist at Pacific Northwest National Laboratory. He is the Principal Investigator for Performance Evaluation on the Cray XMT and for the Energy Smart Data Center project. Dr. Marquez is a high-performance computer and compiler architect who has worked on the development of the German Supercomputers SUPRENUM and MANNA, on the GENESIS European Supercomputer design studies, on US Supercomputer design studies for the Hybrid Technology and Multithreaded Technology (HTMT) computer and on academic high-performance computing projects such as the Efficient Architecture for Running Threads (EARTH) and the Compiler Aided Reorder Engine (CARE). He has published over 30 peer-reviewed papers in journals and conferences in the fields of hardware-, software- and systems architecture and IT infrastructure. He can be reached at andres.marquez@pnl.gov.

Dr. Dennis J. McFarland received the B.S. and Ph.D. degrees from the University of Kentucky, Lexington, in 1971 and 1978, respectively. He is currently a Research Scientist with the Wadsworth Center for Laboratories and Research, New York State Department of Health, Albany. His research interests include the development of EEG-based communication and auditory perception. He can be reached at mcfarlan@wadsworth.org.

Dr. Matthew E. Monroe is a Senior Research Scientist at the Department of Energy's Pacific Northwest National Laboratory working in the area of proteomics. His research involves development of algorithms and software for custom and automated analysis of proteomics and bioinformatics data, including managing, summarizing, and mining the large volumes of data generated by mass spectrometry-based proteomic analysis. Areas of expertise include microcolumn liquid chromatography, LC-MS/MS of peptides and proteins using Q-TOF and ion trap mass spectrometry, and analytical instrumentation design and automation. Dr. Monroe is currently the lead developer for administering, optimizing, and expanding a large collection of SQL Server databases used to process and organize data from the AMT tag process.

Prof. Hausi A. Müller is an Associate Dean Research, Faculty of Engineering and Professor of Computer Science, University of Victoria, British Columbia, Canada. He was founding Director of the Bachelor of Software Engineering program. In collaboration with IBM, CA and SEI, his research group investigates methods, models, architectures, and techniques for self-managing and self-adaptive software-intensive systems. Dr. Müller's research interests include software engineering, software evolution, autonomic computing, diagnostics, SOA governance, software architecture, software reverse engineering, reengineering, program comprehension, visualization, and software engineering tool evaluation. He is Program Co-Chair for IBM CASCON 2010. He was General Chair of VISSOFT 2009 and co-organizer of SEAMS 2006-10 and DEAS 2005. He was General Chair for IWPC 2003 and ICSE 2001. Dr. Müller serves on the Editorial Board of IEEE Transactions on Software Engineering and as Vice-Chair of IEEE Computer Society Technical Council on Software Engineering (TCSE). His e-mail address is hausi@cs.uvic.ca.

Dr. Christopher S. Oehmen is a Senior Research Scientist at the Department of Energy's Pacific Northwest National Laboratory. He is in the Computational Biology and Bioinformatics group focusing on high-performance applications in bioinformatics and machine learning and is the originator of ScalaBLAST, a massively parallel sequence analysis program. Dr. Oehmen earned his Ph.D. in Biomedical Engineering from the Joint Graduate Program in Biomedical Engineering, The University of Memphis and The University of Tennessee Health Science.

Dr. William A. Pike is a Senior Research Scientist at the Pacific Northwest National Laboratory specializing in analytic reasoning, visualization, knowledge representation, and collaborative systems. He is the research coordinator for the Department of Homeland Security's National Visualization and Analytics Center, and also leads work in visualization for cyber security. He has published in such outlets as the *International Journal of Human–Computer Studies*, the *Proceedings and the National Academy of Sciences*, and *Information Visualization*. Dr. Pike holds a Ph.D from Penn State University.

Chad Scherrer is a Research Scientist at the Pacific Northwest National Laboratory whose work involves solving a variety of problems in statistics, mathematical modeling, and high-performance computing. He has contributed to domain problems in aviation safety, cyber security, and radiation detection. Chad leads the Statistical Text Analysis task of the Center for Adaptive Supercomputing Software-Multithreaded Architectures (CASS-MT), which focuses on multithreaded architectures for problems where irregularity leads to load-balance problems or other inefficiencies on conventional hardware.

Dr. Anuj R. Shah is a Senior Research Scientist in the Scientific Data Management group at the Pacific Northwest National Laboratory (PNNL). His expertise is in the field of bioinformatics, heterogeneous data integration, machine learning, high-performance computing, software architectures, and algorithm development. He is the principal investigator on a PNNL Laboratory Directed Research and Development project under the Data-Intensive Computing Initiative on devising high-performance data analysis pipelines for streaming data. His current work includes deisotoping algorithms for proteomics data, application of machine learning frameworks to predict various properties of peptides identified using mass spectrometry and proteomics. Dr. Shah has more than 6 years of experience working in interdisciplinary teams facilitating biological research and has authored and coauthored a number of peer-reviewed journal and conference papers.

Dr. Oreste Villa received his M.S. degree in Electronic Engineering in 2003 from the University of Cagliari (Italy) with final score 110/110. His thesis focused on the design of reconfigurable low-power coprocessor devices. In 2004 he received his M.E. degree in Embedded Systems Design from the University of Lugano (Switzerland) majoring on cycle-accurate multicore simulation frameworks. In 2004, founded by STMicroelectronics, he enrolled the Ph.D. program of Politecnico di Milano (Italy) focusing his research in design and programmability issues of multicore architectures. In May 2006 he joined PNNL (Pacific Northwest National Laboratory) as Ph.D. intern researching on programming techniques and algorithms

for advanced multicore architectures, on cluster fault tolerance and virtualization techniques for HPC. In March 2008 he received his Ph.D. degree from Politecnico di Milano (Italy). Since May 2008, Dr. Villa is a Research Scientist in the HPC group at PNNL, where he focuses his research on performance/programmability of advanced multicore architectures for scientific computing, on architectural studies of advanced architectures for irregular applications, and on cluster fault tolerance.

Dr. Bobbie-Jo Webb-Robertson is a Senior Research Scientist in the computational biology and bioinformatics group at the Department of Energy's Pacific Northwest National Laboratory working in the area of statistical methods development for the analysis of high-throughput data. She has applied machine learning applications to areas such as bioremediation, homeland security, and biomedical research. She is an adjunct Professor of Computer Science at Washington State University and earned her Ph.D. in 2002 in Decision Sciences and Engineering Systems from Rensselaer Polytechnic Institute.

Dr. Paul D. Whitney received his Ph.D. in Statistics from the University of Wisconsin in 1984. He has been a Research Scientist at PNNL since 1991. He has developed information retrieval methods, exploratory analyses algorithms and software for these areas of interest, notably for image and text data. He has both led and contributed to the development of a variety of information visualization methods. His recent research focuses social and behavioral modeling. This research combines data analysis with models for organizations and individual behavior. Dr. Whitney's involvement in this work includes managing projects, developing and implementing low-level algorithms for the characterization and analysis of graph data, leading projects to design transaction analysis tools, and designing/analyzing experiments to assess the effectiveness of the scenario analysis algorithms.

Dr. Jonathan R. Wolpaw, M.D., is Chief of the Laboratory of Neural Injury and Repair at the Wadsworth Center of the NYS Department of Health and a Professor of Biomedical Sciences at SUNY (Albany). His laboratory has developed and is using operant conditioning of spinal reflexes to define the plasticity underlying learning. This work has demonstrated that reflex conditioning changes the spinal cord and has begun to reveal the mechanisms of change, and is now showing that conditioning can guide spinal cord plasticity to improve walking after spinal cord injuries. Dr. Wolpaw is also leading development of EEG-based brain–computer interface (BCI) technology to restore communication and control to people who are paralyzed. His group has shown that noninvasive EEG-based BCI technology can give control similar to that achieved by electrodes placed in the brain, and has

begun to provide BCI systems to severely disabled people for daily use in their homes. These achievements have received wide recognition and numerous national and international awards. He can be reached at wolpaw@wadsworth.org.

Nino Zuljevic is a Software Engineer at the Department of Energy's Pacific Northwest National Laboratory. His primary focus areas are computational applications in homeland security, energy conservation, biology, and risk sciences. He earned his B.S. in Computer Science from Washington State University.

Preface

The *Advances in Computers* has been chronicling the developments in the computer industry since its first volume, published in 1960. This present volume is number 79 in the series. Each year we publish three volumes, each containing five to seven chapters, on new technology affecting the information technology industry. In this current volume, we present five chapters on topics relevant to today's computer user and researcher. The *Advances* is the oldest continuously published book series that keeps up with the every changing face of computer technology.

In volumes 72 (from 2008) and 75 (from 2009), we presented several chapters on the technology useful for the development of new high-performance computers. In Chapter 1 of this present volume, ''Applications in Data-Intensive Computing'' by Anuj R. Shah, Joshua N. Adkins, Douglas J. Baxter, William R. Cannon, Daniel G. Chavarria-Miranda, Sutanay Choudhury, Ian Gorton, Deborah K. Gracio, Todd D. Halter, Navdeep D. Jaitly, John R. Johnson, Richard T. Kouzes, Matthew C. Macduff, Andres Marquez, Matthew E. Monroe, Christopher S. Oehmen, William A. Pike, Chad Scherrer, Oreste Villa, Bobbie-Jo Webb-Robertson, Paul D. Whitney, and Nino Zuljevic, the authors look at data-intensive problems and give several examples of how these new high-speed machines can be used to solve such problems. By data-intensive, they are looking at organizations that capture information at terabytes (or a 1 followed by 12 zeros) of information per day. They first describe the hardware now being developed to process such data and then describe their MeDICi middleware (e.g., a software framework) useful for characterizing and solving such data-intensive applications.

Related to the technology described in Chapter 1, Ami Marowka, in Chapter 2's ''Pitfalls and Issues of Manycore Programming,'' looks at the emerging technology of manycore programming. Computer processors cannot increase in speed arbitrarily. Issues such as the speed of light (and speed of electrons in wires) provide ultimate barriers to increasing processor performance. In order to provide increased performance, modern CPUs contain multiple processors on the same CPU chip. Thus, new PCs now contain two, four, or more independent processors, each running at the same clock rate as earlier single processor machines. The difficult problem is how to use all processors effectively so that applications can make better use of all

this computing power. Previously this was the domain of the large supercomputers running at well-funded research laboratories. Now it is a problem faced by almost every user of a modern desktop or laptop machine. How to understand and address the software and hardware limitations of such multicore processors is the goal of this chapter.

Alfred Loo, in Chapter 3's ''Illusion of Wireless Security,'' addresses the problem of providing security in a wireless world. Today, an increasing percent of users connect to the Internet via wireless connections—a local area network (LAN) in their home or place of employment or a regional wireless network (WIFI). How secure are these connections since wireless signals are easy to intercept and often easy to interpret? What security mechanisms are in place, how secure are they, and what improvements can be made to add to their security? These related topics are the focus of this chapter.

In Chapter 4, Dennis Mcfarland and Jonathan Wolpaw, in ''Brain–Computer Interfaces of the Operation of Robotic and Prosthetic Devices,'' describe an exciting new development in computer technology. Since the human brain produces signals that are detectable on the scalp, brain–computer interfaces can translate these signals into outputs that communicate a user's intent without the participation of peripheral nerves and muscles. This allows for the direct control of devices without the need for invasive surgery. Especially for those with physical limitations and handicaps, such devices offer the ability to replace biological functions with mechanical ones. The authors explore this topic further.

In the last chapter, ''The Tools Perspective on Software Reverse Engineering: Requirements, Construction and Evaluation'' by Holger M. Kienle and Hausi A. Müller, the authors provide a comprehensive discussion of the problems in reverse engineering software. It is often necessary to understand the designs of a software system when you only have the executable program. The design may have been lost or the design documents might not have kept up with the changing role of the program itself as it became modified over numerous versions and over many years. The authors address the reverse engineering task from three perspectives: (1) requirements for reverse engineering tools, (2) construction of reverse engineering tools, and (3) evaluation of reverse engineering tools.

I hope that you find these chapters of interest. I am always looking for new and interesting topics to appear in these pages. If you have any suggestions of topics for future chapters, or if you wish to contribute such a chapter yourself, I can be reached at mvz@cs.umd.edu.

Marvin Zelkowitz
College Park, Maryland

Applications in Data-Intensive Computing

ANUJ R. SHAH

Pacific Northwest National Laboratory, Richland, Washington, USA

JOSHUA N. ADKINS

Pacific Northwest National Laboratory, Richland, Washington, USA

DOUGLAS J. BAXTER

Pacific Northwest National Laboratory, Richland, Washington, USA

WILLIAM R. CANNON

Pacific Northwest National Laboratory, Richland, Washington, USA

DANIEL G. CHAVARRIA-MIRANDA

Pacific Northwest National Laboratory, Richland, Washington, USA

SUTANAY CHOUDHURY

Pacific Northwest National Laboratory, Richland, Washington, USA

ADVANCES IN COMPUTERS, VOL. 79
ISSN: 0065-2458/DOI: 10.1016/S0065-2458(10)79001-X

IAN GORTON

Pacific Northwest National Laboratory, Richland, Washington, USA

DEBORAH K. GRACIO

Pacific Northwest National Laboratory, Richland, Washington, USA

TODD D. HALTER

Pacific Northwest National Laboratory, Richland, Washington, USA

NAVDEEP D. JAITLY

Pacific Northwest National Laboratory, Richland, Washington, USA

JOHN R. JOHNSON

Pacific Northwest National Laboratory, Richland, Washington, USA

RICHARD T. KOUZES

Pacific Northwest National Laboratory, Richland, Washington, USA

MATTHEW C. MACDUFF

Pacific Northwest National Laboratory, Richland, Washington, USA

ANDRES MARQUEZ

Pacific Northwest National Laboratory, Richland, Washington, USA

MATTHEW E. MONROE

Pacific Northwest National Laboratory, Richland, Washington, USA

CHRISTOPHER S. OEHMEN

Pacific Northwest National Laboratory, Richland, Washington, USA

WILLIAM A. PIKE

Pacific Northwest National Laboratory, Richland, Washington, USA

CHAD SCHERRER

Pacific Northwest National Laboratory, Richland, Washington, USA

ORESTE VILLA

Pacific Northwest National Laboratory, Richland, Washington, USA

BOBBIE-JO WEBB-ROBERTSON

Pacific Northwest National Laboratory, Richland, Washington, USA

PAUL D. WHITNEY

Pacific Northwest National Laboratory, Richland, Washington, USA

NINO ZULJEVIC

Pacific Northwest National Laboratory, Richland, Washington, USA

Abstract

The total quantity of digital information in the world is growing at an alarming rate. Scientists and engineers are contributing heavily to this data ''tsunami'' by gathering data using computing and instrumentation at incredible rates. As data volumes and complexity grow, it is increasingly arduous to extract valuable information from the data and derive knowledge from that data. Addressing these demands of ever-growing data volumes and complexity requires game-changing advances in software, hardware, and algorithms. Solution technologies also must scale to handle the increased data collection and processing rates and simultaneously accelerate timely and effective analysis results. This need for ever faster data processing and manipulation as well as algorithms that scale to high-volume data sets have given birth to a new paradigm or discipline known as ''data-intensive computing.'' In this chapter, we define data-intensive computing, identify the challenges of massive data, outline solutions for hardware, software, and analytics, and discuss a number of applications in the areas of biology, cyber security, and atmospheric research.

1. Introduction 4
2. What Is Data-Intensive Computing? 6
   2.1. Hardware Architectures 7
   2.2. Data-Intensive Data Analytics 13
   2.3. Software Infrastructures for Data-Intensive Applications 15
3. Applications in Data-Intensive Computing at PNNL 20
   3.1. Applications in Biological Sciences 21
   3.2. Data-Intensive Cyber Analytics 39
   3.3. Applications in Atmospheric Radiation Measurement Program 49
4. Conclusions 62
   Acknowledgement 63
   References 63

1. Introduction

Advances in technology have empowered individuals with the ability to generate digital content with mouse clicks and voice commands. Digital pictures, e-mails, text messages, home videos, audio, and Web pages are common examples of digital

content that are generated on a regular basis. A family on vacation can generate pictures and video content that require up to several terabytes of disk space. According to a comprehensive study [1] by IDC, a provider of market intelligence, individuals generate 70% of the currently existing digital content. The total amount of information in the world in 2006 was estimated at about 160 exabytes and is expected to grow to 1 atabytes by the end of 2010, amounting to a sixfold increase in data volume. In terms of size, 1 atabyte equates to a stack of books that could reach from the sun to Pluto and back. A recent update to the 2007 Gantz study [2] estimates that the digital universe is 10% bigger than originally thought in 2007. The 2008 report identifies the following rapidly growing areas of the digital universe: digital television, surveillance cameras, Internet access in emerging countries, sensor-based applications, data centers supporting ''cloud computing,'' and social networks.

The Gantz study therefore alludes to the fact that scientists and engineers are quickly becoming significant contributors to this data ''tsunami.'' Scientists and engineers engaged in research are generating and gathering data at incredible rates. For example, Chevron Corporation accumulates data at a rate of 2 TB a day, or 23 MB every second [3]. In science, high-energy physics remains the leading generator of raw data. The Atlas experiment for the Large Hadron Collider (http://atlasexperiment.org) at the Center for European Nuclear Research generates 1 billion proton collision events per second. If all the data were recorded, it would fill 100,000 compact disks every second. Despite the immense scale of this experimental facility, it is not the only colossal science project of recent years [4]. The sequencing of the human genome, a milestone in the field of biology, has opened the floodgates for several follow-up studies that aim to explore the complex workings of the cell. In astronomy, data collection now exceeds the terabyte range on a nightly basis [5] and the amount of data collected will grow incredibly as new high-resolution telescopes are deployed in the next decade. In climate modeling, currently used models generate terabytes of data [6] and future models promise petabytes of data available for analysis as the resolution of climate simulations improve.

Sheer data volume is only part of the problem. The complex nature of these data sets introduces an additional level of processing challenge. For example, the Atlas experiment is a one-of-a-kind effort to understand the nature of basic forces, and the information gained may change our children's science textbooks. It brings experimental physics into new territory and will force scientists to look at new ways of analyzing and understanding the large volumes of data generated. Data complexity also is an attribute of most biological experiments and studies. For example, an effort to characterize viruses in aquatic microorganisms involved the collection of ocean samples from the North Atlantic, the Panama Canal, and the South Pacific [7]. This effort yielded a total of 7.7 million sequencing reads. Understanding the

functional characteristics of the cell involves detailed modeling and simulation of individual processes arising from complex interactions between the numerous constituents of a cell such as proteins, DNA, RNA, and small molecules [8]. As another example, climate modeling presents numerous challenges in describing the details of the atmosphere, the oceans, the land, as well as the energy from the sun. Scientists use these models to make predictions hundreds of years into the future.

Revolutions in scientific experimentation, data sensor diversity, computing power, and the availability of inexpensive, distributed communications have driven this explosion in the volume and complexity of data. As data volumes and complexity grow, it is increasingly arduous to extract valuable information from the data and derive knowledge. Scientists spend many computer hours trying to organize, sort, and filter the data to manageable pieces so they can draw conclusions about the big picture [9]. They follow prescribed procedures for data filtering, processing, and analysis to reach their conclusions.

The reductionist approach to data analysis has provided invaluable knowledge for centuries and has allowed scientists to detect the unexpected. However, as data volumes grow and analysis becomes more complex, discovering new information and extracting knowledge becomes more difficult [10]. Therefore, progress in science and discovery will be achieved only by effectively coupling computational power with experimental and field data through innovations in data and knowledge management, information analytics, visualizations, and decision analysis. Addressing these demands of ever-growing data volumes and complexity requires game-changing advances in software, hardware, and algorithm development. Solution technologies also must scale to handle the increased data collection and processing rates and simultaneously accelerate timely and effective analysis results. This need for faster data processing and manipulation as well as algorithms that scale to high-volume data sets have given birth to a new paradigm or discipline known as ''data-intensive computing'' (DIC).

In Section 2, we tackle the definition of DIC. In Section 3, we discuss applications of DIC through explicit examples of three areas of work at Pacific Northwest National Laboratory (PNNL): biology, cyber security, and atmospheric research.

2. What Is Data-Intensive Computing?

We define DIC as ''collecting, managing, analyzing, and understanding data at volumes and rates that push the frontiers of current technologies'' [11]. One can treat DIC as a computational paradigm in which the sheer volume of data is the dominant performance parameter. DIC promises not only an evolutionary change in

informatics, but also revolutionary changes in the way researchers gather and process information. The impacts of these changes will range from the hardware and algorithms to the presentation of knowledge to the end user.

DIC facilitates human understanding of complex problems. Data-intensive applications provide timely and meaningful analytical results in response to exponentially growing data complexity and associated analysis requirements through the development of new classes of software, algorithms, and hardware.

To be more specific about what DIC encompasses, it is valuable to determine how to quantitatively define the term. But, quantifying the meaning of DIC is heavily impacted by the complexity of factors that need to be considered, and quantification is limited to a point in time since DIC is rapidly evolving. Classifying a problem as data intensive could depend upon the data rates (gigabytes/s to terabytes/s) and data volumes (terabytes to petabytes) involved, but other factors such as the variability in data rate, bandwidth of data paths, number of data-handling units, complexity of the data and analysis, and human limitations in interacting with the data can all be important. In the following sections, we describe the state of the art in hardware architectures, data analytics, and software application architectures that are key ingredients in DIC applications.

2.1 Hardware Architectures

Data-intensive applications present unique challenges for computing systems and architectures. Applications can be roughly divided into two categories: irregular and regular, based upon memory access patterns. Regular applications, such as aircraft modeling, climate modeling, and molecular dynamics, typically are driven by computational kernels that have good spatial and temporal locality (e.g., dense matrix multiplication, structured grids) and can benefit from hierarchical memory structures and caches. Irregular applications, such as social network and media analysis, cyber network analysis, and infrastructure (electrical, transportation, communication) modeling, typically have computational kernels based on graph algorithms (e.g., centrality, breadth-first search, connected components) that have poor locality and cannot benefit from cache-based systems.

With the explosion of data and data-intensive applications over the past decade, there has been an accelerated increase in novel computing architectures. These range from computationally intense, data-parallel streaming processors such as graphics processing units (GPUs), customizable logic in field-programmable gate arrays (FPGAs), massively multithreaded processors that can hide memory latency, active-storage-based clusters where processors are housed on disks to minimize data movement yet scale in computational capability as more active disks are added, to solid-state disk technology that is dramatically increasing disk performance.

Over the past few years, researchers have actively investigated how to exploit this heterogeneous pool of new architectures to meet the rapidly growing challenges of both regular and irregular data-intensive applications.

2.1.1 Graphics Processing Units

A 2007 Price Waterhouse Coopers report [12] projected that the international market for video games would grow to $48.9 billion dollars by 2011. This followed a 2006 report from the Entertainment Software Association [13], in which it was noted that the US global sales of entertainment software in 2004 were already $25.4 billion. In 2004, the leading providers of graphics coprocessors (nVidia and ATI) had just started to provide more flexible programming access to their special purpose hardware. The $25 billion dollar market that was thriving in 2004 drove, and continues to drive, a highly specialized computing hardware market that has produced revolutionary processor advances, with new generation systems introduced every 6–12 months—targeted specifically to accelerating video game performance. These processors employ hundreds of floating-point units and operate in a streaming single instruction multiple data (SIMD) mode.

With the introduction of increased programmability, there were several initial studies on how to exploit GPUs for more general applications beyond the gaming market. For example, shortly after the first programmable GPUs entered the market, Flath et al. [14] successfully fielded a real-time aircraft-based system that performed georegistration of streaming high-resolution imagery (including jitter removal, motion detection, and blob finding) at 44 Mpixels/s on up to 5 TB of raw data enabling real-time transmission of high-resolution imagery over low bandwidth radio communication. Johnson et al. [15] demonstrated the use of GPUs for a range of problems in image processing and knowledge discovery with speedups over conventional processing architectures of up to two orders of magnitude. They also were the first to emulate double-precision arithmetic in software on GPUs while still realizing performance gains over traditional CPUs. Luebke et al. [16] demonstrated searching and sorting as well as some database operations on GPUs. Techniques for programming GPUs were introduced in Ref. [17]. It was quickly recognized that GPUs can offer orders of magnitude performance increases for a wide range of applications that can leverage the massive investment in computing gaming (see Ref. [18] for a survey). In 2008, nVidia announced the T10P Tesla GPU with 240 cores that can achieve a teraFLOP[1] of (single precision) computing power, thus

[1] A FLOP is 1 floating-point operation per second.

solidifying the role of the GPU as a driving architecture for high-performance DIC. To meet the ever expanding mobile market, nVidia has recently announced the Tegra processor—a system-on-a-chip integrating CPU, GPU, and memory controller that can be used on handhelds. At this point GPUs have entered the mainstream for DIC with market pressure leading the introduction of new software development frameworks for GPUs such as CUDA—a C language-based parallel architecture that simplifies the programming of GPUs [19]. GPUs and GPU clusters have been used for many general-purpose data-intensive applications ranging from real-time image and speech processing [20], to document analysis [21], database operations [22], and bioinformatics [23].

While not strictly a GPU, the Sony Toshiba IBM (STI) Cell processor was introduced to provide accelerated coprocessing for the Sony PlayStation 3 [24, 25]. While GPUs have hundreds of parallel SIMD cores, the STI Cell has eight ''synergistic processing elements,'' connected via a high-speed bus and a controller PowerPC processor. The SPEs can operate as SIMD processors, controlled by a PowerPC but they can also pass messages and data to each other over the high-speed bus. The STI Cell processor offers more direct control of data movement and algorithm flow than does the GPU and it has been demonstrated to perform over 200 GFLOPs. The STI Cell has been utilized for data-intensive problems in scientific computing [26], biology [27], speech processing [28], string matching [29], graph algorithms [30], multimedia indexing [31], and many other areas.

2.1.2 Reconfigurable Computing Devices: Field-Programmable Gate Arrays

Many fields in experimental science are moving toward high-throughput methodologies where scientific instruments can process large numbers of samples and produce corresponding massive amounts of data. For this reason, it is becoming increasingly important to perform online processing of the data as it is being produced by the instrument, without having to wait for data to be captured and then processed offline. While GPUs and general-purpose CPUs can be used for this processing, often there is specialized domain-specific logic that is needed to achieve adequate performance. FPGA boards offer the opportunity for the application developer to create specialized, reconfigurable hardware that can be directly attached to the data capture ports of a digital instrument to provide real-time data processing and analysis. Recent work has demonstrated that FPGAs can provide real-time complex data processing in many fields including text categorization [32],

image processing [33], DNA sequencing [34], network intrusion detection [35], real-time delivery of financial information [36], and other applications.

Developing and testing FPGA designs that will be processing streaming data from an instrument requires a significant design, development, and testing efforts. It is also quite difficult to determine beforehand what footprint a particular processing design will occupy and what its performance will be. Fortunately more flexible and less demanding platforms for initial prototyping and testing can be employed. Hybrid CPU/FPGA high-performance computing (HPC) systems provide a platform to develop and prototype processing algorithms for FPGAs that can be tested under more flexible software control while still having to deal with details, nuances, and idiosyncrasies of hardware platforms.

2.1.3 Multithreaded Systems

The increasing performance differential between the capabilities of memory subsystems and microprocessors has caused a large class of applications to become memory-bound: that is, their performance is determined mainly by the speed at which the memory subsystem can deliver data words to the microprocessor. Over the years, several hardware and software mechanisms have been proposed to increase the performance of such applications by reducing the exposed stall times seen by the microprocessor. Most commodity microprocessors utilize a cache hierarchy, whereby small sections of high-speed memory hold data, which has been recently fetched from main memory. Cache mechanisms are highly effective for regular applications that exhibit good temporal and spatial locality. However, many irregular applications do not exhibit such locality; their memory access patterns are not easily predictable. This is particularly true for data-intensive applications that use dynamically allocated large, pointer-linked data structures such as graphs and trees.

Multithreaded architectures maintain multiple threads of execution and utilize hardware-based context switching to overlap the memory latency incurred by any thread with the computations from other threads. The Cray MTA-2 processor [37] and its successor the Cray Threadstorm processor [38] in the single job acceleration category and Sun Niagara-1 [39] and Niagara-2 [40] processors in the throughput category are examples of state-of-the-art multithreaded processors. These processors represent very different approaches to realizing higher application performance through the use of multiple thread contexts to overlap memory access latencies. Because of the memory latency tolerance, these multithreaded platforms have the potential of significantly improving the execution speed of irregular data-intensive applications.

2.1.3.1 Cray MTA-2 and XMT. The Cray MTA-2 processor exploits thread-level parallelism by interleaving 128 hardware thread streams on one instruction pipeline [41,42]. Each stream has a 32-entry register file, a status word and a program counter associated with it. Ready stream selection to issue instructions does not incur cycle penalties. A long instruction word is comprised of a memory operation, a fused multiply–add, and a branch/logical or floating-point add operation. The MTA-2 supports fine grain synchronization by guarding each memory word with full/empty bits.

The Cray XMT is the commercial name for the shared-memory multithreaded machine developed by Cray [43] under the code name ''ELDORADO'' [38,41,42]. The system is composed of dual-socket Opteron AMD service nodes and custom-designed multithreaded compute nodes with Threadstorm processors. The entire system is connected with the Cray Seastar-2.2 high-speed interconnect. The XMT system can scale up to 8192 Threadstorm processors and 128 TB of shared memory. Each Threadstorm is associated with a memory system that can accommodate up to 8 GB of 128-bit wide DDR memory. Each memory controller has an associated 128 KB buffer. Memory is structured with full–empty-, pointer forwarding-, and trap-bits to support fine-grained thread synchronization with little overhead. The memory is hashed at a granularity of 64 bytes (Fig. 1) and fully accessible through load/store operations to any Threadstorm processor connected to the Seastar-2.2 network, which is configured in a 3D toroidal topology. While memory is

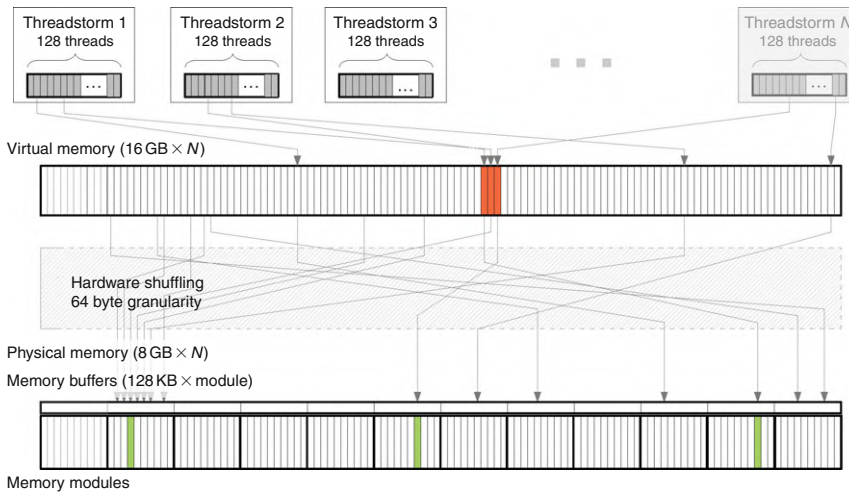

FIG. 1. Cray XMT memory access structure.

completely shared among Threadstorm processors, it is decoupled from the main memory in the AMD Opteron service nodes. Communication between Threadstorm nodes and Opteron nodes is performed through a Lightweight Communication Library (LUC). Continuous random accesses to local memory by the Threadstorm processor caps memory bandwidth at around 100 million requests per second, whereas global access can sustain request rates of 90–30 million with system configurations ranging from 1000 to 4000 processors.

The software environment on the Cray XMT includes a custom, multithreaded operating system for the Threadstorm compute nodes, a parallelizing C/C++ cross-compiler targeting Threadstorm, a standard Linux 64-bit environment executing on the service and I/O nodes, as well as the necessary libraries to provide communication and interaction between the two parts of the XMT system. The parallelizing C/C++ compiler generates multithreaded code that is mapped to the threaded capabilities of the processors automatically. Parallelism discovery happens fully or semiautomatically by the addition of pragmas (directives) to the C/C++ source code. The compiler parallelism discovery focuses on analyzing loop nests and mapping loop iterations in a data-parallel manner to threads.

2.1.4 Active-Storage Systems

The notion of active storage and its application to DIC was proposed over a decade ago by Riedel et al. [44]. The principle of active storage is to move portions of the application-level code to processing units on the disk. This idea has been extended and rapidly commercialized through systems, such as the Netezza data warehouse, that integrate FPGAs and general-purpose processors at the hardware level with the disk to provide computational capability at the disk so that computations can be distributed and performed where the data reside, rather than moving large amounts of data across a network. These systems have been demonstrated to offer orders of magnitude performance increases for problems in graph search and natural language understanding. Yoo et al. [45] demonstrated the largest known bidirectional breadth-first search on a 300 billion edge graph, with 95% of the searches completing in less than 5 min, using a Netezza data warehouse. Also using a Netezza system, Davidson et al. [46] studied publication citation graphs, a key component in the field of bibliometrics. They were able to analyze bibliographic coupling (where two papers cite a common earlier reference) up to 46 billion coupling coefficients and achieved two orders of magnitude speedup over traditional architectures. This study also explored implementations of graph search on integrated circuit network data and word sense disambiguation in natural language processing.

2.1.5 Solid-State Storage

Solid-state devices have been studied for decades, where ''solid state'' in this context is to be distinguished from ''mechanical'' or spinning-disk technology. Solid-state devices, composed of NAND flash nonvolatile memory, have traditionally been financially prohibitive for use as general-purpose storage devices. However, the rapid advances in DIC—the need for people to exchange music, pictures, movies, and share files—led to the development of USB flash drives, and are now requiring advanced disk technologies. Solid-state drives developed by FusionIO have recently been demonstrated to provide significantly increased speed over spinning disk [47]. One of the more interesting results in the recent investigation of solid-state storage was that increasing the number of I/O threads increased the performance of data transfers, contrary to the behavior of current spinning-disk drives [48]. Ajwani et al. [49] also provide an initial evaluation of solid-state storage devices and conclude that flash drives outperform traditional hard drives for read operations, but not for write operations. However, they also note that existing external memory models for algorithm design that are based on traditional storage architectures do not fully exploit the capability of solid-state devices and that more work needs to be done to better design algorithms to take advantage of these new technologies. Results like these have driven new approaches to data-intensive application organization and design that may seem counterintuitive given previous generations of systems. Technical performance differentiators of solid-state technologies compared to traditional spinning disk and the growth in commercial viability of these systems are likely to have significant implications on data-intensive applications in the next few years.

2.2 Data-Intensive Data Analytics

The conventional approach to data analysis is a sequential process in which data are first generated and stored to be processed later. However, in our ever-changing digital landscape, postponing the data analysis can have significant impacts on the rate of scientific and knowledge discovery. The rate at which digital information is generated is making it imperative to invent new algorithms and data processing mechanisms that are capable of analyzing streaming and high-volume data.

The challenges associated with the scale and complexity of high-volume streaming data have both computational and human factors perspectives. One challenge is that the complexity of the information in the data increase with the data set size. Accordingly, the representations used in analysis must scale similarly. Another challenge is that the complexity of the information in petabyte-scale data dwarfs the comprehension capacity of a human learner. Visual approaches help address this challenge.

2.2.1 Objectives and Approach

Data analysis ends with human understanding and action. However, the information scale dwarfs individual capabilities for understanding and is often compared to trying to take a drink from a fire hose. In such situations, the sheer data volume makes data discovery incredibly difficult.

Many analysis challenges also stem from the heterogeneity of the data that must be processed. For instance, for cyber security, the network packets, various system states, and contextual information (e.g., collaborators, news releases, security/hacker Web sites, etc.) are all relevant for indicating the current and future health of the nodes on a network. In such applications, data types and objects are numerous and vast in size. For example, network traffic for 1 year is in the exabyte range. Modern scientific instrumentation is capable of creating terabytes of structured data on a daily basis. Computer-based simulations generate information at large scales. Digital imagery is easy to create and store.

The diversity of data types encountered can be addressed by a common strategy:

1. The data objects, be they images, text, network packet, or numeric, are encoded as numeric or categorical vectors.
2. The vectors then are analyzed with available data analysis routines.
3. The analytic results are mapped back into the original domain of the data.

The advantage of this strategy is that it allows an analyst to rapidly explore diverse data types with standard data analytic methodologies and algorithms. Once the first step is completed, the wealth of available software for clustering, classifying, trending, and relating numeric data can be mapped to the analysis of text or network or other data objects. The individual coordinates in the numeric vector are not necessarily meaningful. Of course, to be useful, the entire vector needs to discriminate with respect to the decision problem(s) at hand. There is extensive experience in diverse technical communities in accomplishing the above sequence of analytic steps. For text, this experience includes n-grams, word frequencies, mappings into concept spaces, such as word-net and dimension-reduced forms [50–52]. For networks, there are standard approaches based on graph characteristics [53]. Once the vector data are analyzed, the results are mapped back into the context of the data. So, document clusters that are calculated based on vectors that represent each document can be labeled with text from the documents in the clusters. See Whitney et al. [54] for additional examples.

Frameworks for understanding and discussing the scaling of data analysis algorithms and computing are established (e.g., [53,55]). Wegman [55] suggests that, for purposes of scaling analyses to data analysis problems, it is necessary to employ

algorithms that have complexity at most $O(n)$, where n scales as the number of floating-point data items.

For addressing large data stream problems, there are standard categories of approaches. These categories are described below:

- *Analytic focus*. A typical instance in the context of the Internet is retrieval by key word search. This is, from the perspective of a user, a simple act that greatly reduces the amount of available information to a manageable amount. Another instance in this category of analyses is anomaly detection, which focuses user attention on atypical instances within the data stream.
- *Summarization*. In this category, high-level views of the data are created. A cluster analysis is a standard instance of a data summary where the data stream is summarized as the occurrence of a list of clusters. Machine learning results in a focused listing of categories that occur. There is the potential for significant loss of information. However, in conjunction with capabilities that use the summary as an entry point into the data, summarization is quite powerful.
- *Data reduction*. This is more often an intermediate step as opposed to a direct analytic output. The feature calculation used to create the vectors in the analysis pattern can greatly reduce the size of the data stream. These features are often ''lossy,'' but are designed with the intent of capturing the information in the data stream.

2.3 Software Infrastructures for Data-Intensive Applications

Data-intensive applications exist in diverse forms. Some simply require an algorithm to run over a massive data set, and often these can be trivially parallelized and executed on a cluster in a single program. Other applications require more complex analytics that are often too computationally expensive to run on complete data sets. In such cases, preprocessing steps are required to reduce the data set to a form that is amenable for complex analytics. Still other applications can operate on very large volumes of reasonably structured data, making them amenable for deployment on parallel database systems that support query languages such as SQL. The following sections explore these alternatives in more detail.

2.3.1 Data Processing Pipelines

Emerging from the scientific domains, many large data problems are addressed with processing pipelines. A pipeline is initiated when raw data that originate from a scientific instrument or a simulation are captured and stored. The first stage of

processing typically applies techniques to reduce the data in size by removing noise, or process it (e.g., index, summarize, or markup) so that it can be more efficiently manipulated by downstream analytics. Once the capture and initial processing has taken place, complex algorithms search and process the data.

These algorithms create information and/or knowledge that can be utilized by individuals or further computational processes. Often, these analytics require large-scale distributed or specialized HPC platforms to execute. Finally, the analysis results are presented to users so that they can be digested and acted upon. This stage may use advanced visualization tools, and enable the user to step back through the processing steps that have been executed in order to perform forensic investigations to validate the outcome. Users also may need facilities to modify parameters on some of the analytics that have been performed and re-execute various steps in the processing pipeline.

As depicted in Fig. 2, processing pipelines start with large data volumes having low information content. These data are reduced by the subsequent processing steps in the pipeline to create relatively small data sets that are rich in information and are suitable for visualization or human understanding. In many applications, for example, the Atlas (http://atlas.web.cern.ch/Atlas/index.html) high-energy physics experiment; large data sets are moved between sites over high-speed, wide-area networks for downstream pipeline processing.

Constructing applications structured as processing pipelines is a complex activity. Pipelines can have complex topologies, incorporating branches, loops, and merges. They also typically comprise distributed codes and require the pipeline to marshal the data sets between each step in the pipeline. Many programmatic approaches can

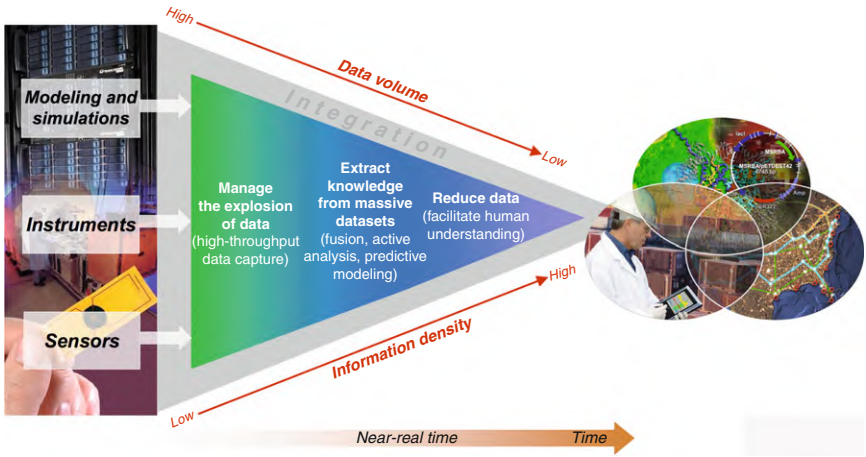

FIG. 2. Blueprint for a data processing pipeline.

be employed to coordinate the various steps in the processing, ranging from operating system-level shell scripts to generic workflow coordination technologies such as MeDICi (middleware for data-intensive computing) [56] and Kepler [57].

2.3.1.1 Data Warehouses.

Commercial enterprises are voracious users of data warehousing technologies. Mainstream database technology vendors supply these technologies to provide archival storage of business transactions for business-analysis purposes. As enterprises capture and store more data, data warehouses have grown to petabyte size. Best known is Wal-Mart's data warehouse, which over the span of a decade has grown to store more than a petabyte of data [58], fueled by daily data from 800 million transactions generated by its 30 million customers.

The data warehousing approach also finds traction in science. The Sloan Digital Sky Survey (SDSS) (http://cas.sdss.org/dr6/en/) SkyServer stores the results of processing raw astronomical data from the SDSS telescope in a data warehouse for subsequent data mining by astronomers. While the SkyServer data warehouse currently only stores terabytes of data, it has been suggested that the fundamental design principles can be leveraged in the design [59] of the data warehouse for the Large Synoptic Survey Telescope (www.lsst.org) that is to commence data production in 2012. The telescope will produce 6 petabytes of raw data each year, requiring the data warehouse to grow at an expected rate of 300 TB/year.

2.3.1.2 Cloud Computing.

Driven by the explosive growth of the Internet, search enterprises such as Google, Microsoft, Amazon, and Yahoo! have developed multipetabyte data centers based on low-cost commodity hardware. Data are stored across a number of widely geographically distributed physical data centers, each of which might contain over 100,000 nodes. Access to the data is provided by a set of services that are available over standard Internet protocols (IPs) such as Web services. These massive data centers and the software infrastructure that developers use to create applications are known as ''clouds.''

Clouds provide the following two main advantages:

1. The illusion of infinite computing resources available on demand, thereby eliminating the need for cloud computing users to invest in expensive computing infrastructures.
2. The elimination of an up-front commitment by cloud users, thereby allowing organizations to scale the use of hardware and software resources to meet their needs.

Programming models such as MapReduce [60] and its open-source counterpart, Hadoop (http://hadoop.apache.org/) provide abstractions to simplify writing applications that access this massively distributed data collection. Essentially, MapReduce distributes data and processing across clusters of commodity computers and processes the data in parallel locally at each node. In this way, massively parallel processing can be simply achieved with clusters that comprise thousands of nodes. In addition, the supporting run-time environment provides transparent fault tolerance by automatically duplicating data across nodes and detecting and restarting computations that fail on a particular node.

For data-intensive applications that are inherently suitable for data-parallelism approaches, cloud computing is an attractive option. Massive data sets can be distributed across the cloud and segmented into chunks that can be processed in parallel by tasks running on individual nodes across the cloud. This effectively provides a batch processing and analytics environment for applications that analyze large data sets. Not surprisingly, this approach is attracting interest from the scientific community. The National Science Foundation is partnering with Google and IBM to provide a 1600-node cluster for academic research (http://www.nsf.gov/news/news_summ.jsp?cntn_id=111186). Supported by the Hadoop open-source software, this provides an experimental platform for scientists and researchers to investigate new DIC applications.

2.3.2 MeDICi—The Middleware for Data-Intensive Computing

A major cause of overheads in data-intensive applications is moving data from one computational resource to another. In an ideal situation, data are produced and analyzed at the same location, making movement of data unnecessary. In reality, however, this is not always possible. For example, scientific instruments such as telescopes, mass spectrometers, or genome-sequencing machines typically generate data locally and the resulting data sets are moved to computational sites for detailed analyses. In cyber security, network traffic is captured and initially processed locally by nodes that act as network sensors. Periodically, the processed data are transmitted to an analytics and data archiving site at which it is combined with data from the whole network and analyzed for suspicious behavior.

For this reason, we have created the MeDICi technology [61]. The MeDICi code is a middleware platform for building complex, high-performance analytical applications. These applications are structured as workflows (also called pipelines) of software components, each of which perform some analysis on incoming data and then pass their results to the next step in the workflow. The MeDICi project

comprises three, loosely coupled subprojects, which are briefly described below and depicted in Fig. 3:

1. *MeDICi integration framework (MIF)* is a component-based, asynchronous messaging platform for distributed component integration.
2. *MeDICi workflow* is a Business Process Execution Language [62]-based design and execution environment that integrates with MIF components or standard Web services to provide workflow definition tools and a standards-based recoverable workflow execution engine.
3. *MeDICi provenance* is a Java API, Resource Description Framework (RDF)-based store and content management system for capturing and querying important metadata that can be used for forensic investigations and reconstruction of application results.

In a MeDICi workflow application, the designer creates a workflow graphically with the DWF language, which is a simplification of the standard BPEL workflow description language. Each task in the workflow calls an associated Web service, which may be a standard service located somewhere on the Internet, or a Web service supported by a MIF component deployed in the MIF container. MIF components wrap computational codes that require complex integration, and support a

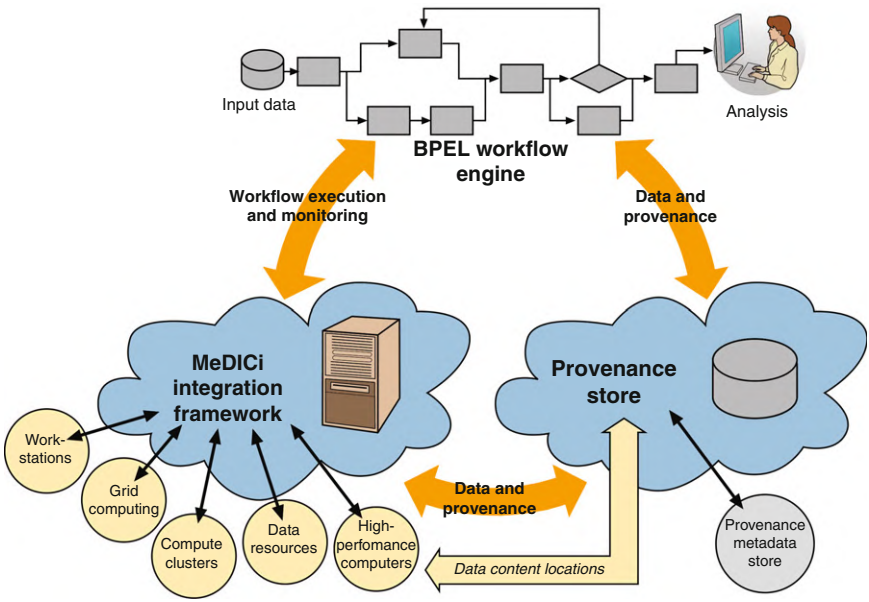

FIG. 3. MeDICi architecture overview.

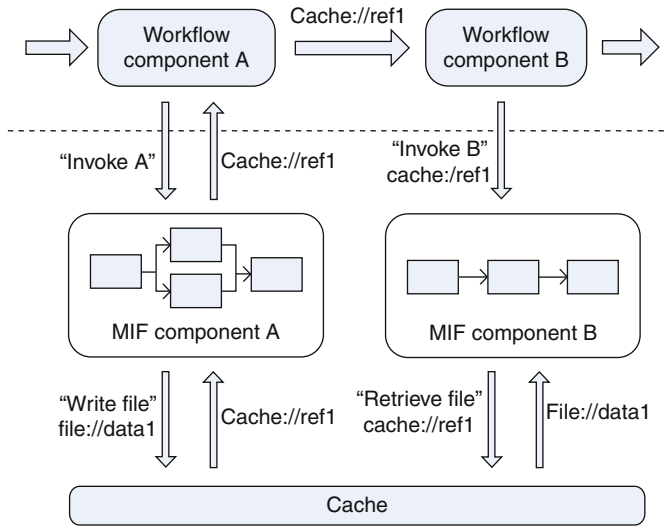

FIG. 4. Efficient data set exchange between components in MeDICi workflow.

protocol that is designed to minimize the data transfer overheads between elements of the workflow. This architecture is shown in Fig. 4.

Optionally, MIF components can record metadata about the data (known as provenance) they receive/produce, and the processing carried out. The metadata are passed transparently from a component to the MIF container, and this sends a message to a message queue called ProvenanceListener. MeDICi provenance takes these messages from the queue, and stores them in an RDF store for subsequent analysis by scientists.

MeDICi is a freely available, open-source technology that can be downloaded from http://medici.pnl.gov. It is being used in applications in atmospheric sciences, cyber security, bioinformatics [63], and carbon sequestration research.

3. Applications in Data-Intensive Computing at PNNL

In the following sections, we discuss applications in DIC from a number of scientific domains that we are working on at PNNL. First, we describe several data-intensive applications we are working on in biology. Then, we discuss

applications in cyber security where the biggest challenges are presented in the fields of data streams and analysis of real-time network traffic. Finally, we discuss data-intensive applications of the Atmospheric Radiation Monitoring Program, which is a Department of Energy (DOE) sponsored, collaborative effort involving a number of research institutes.

3.1 Applications in Biological Sciences

Since commencing the sequencing of the human genome [64], the biological sciences have posed interesting challenges for computing. The sheer volume of genomic and proteomic data generated at individual research institutions has outpaced the development of applications to process the data. In this section, we first discuss a common data-intensive application in biological sequence analysis. The comparison of a large number of biological sequences against a very large database to discern the extent of similarity is a fundamental problem in bioinformatics. Then we discuss data-intensive applications in proteomics. Using high-resolution liquid chromatography (LC) coupled with mass spectrometry as our example, we discuss a novel technology that has broad application in sensor analytics and smart online data collection. We also discuss the application of novel hardware architectures in the context of ion mobility spectrometry.

3.1.1 Biological Sequence Analysis

3.1.1.1 The Challenge. In much the same way that computer components have undergone continuous significant improvement in performance, the technology for determining the linear sequence of molecular units in genes and proteins has transformed the way in which modern biological research is conducted. At one time, determining the sequence of a single gene or protein required heroic effort and led to an environment where laboratories (by necessity) specialized in studying a single gene or class of genes. Today, things are much different. Sequencing technology has improved to the point where obtaining the complete genome for cultured organisms is within the reach of many single-investigator grants, in terms of both expense and time. Small- and large-scale genome-sequencing centers regularly complete and publish newly sequenced genomes, typically doubling the entire volume of sequenced genes every 18 months (http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html)! This has led to an explosion in the public genome information available and a data-driven revolution in the field of microbial biology. More complex eukaryotic organisms such as plants and animals have correspondingly more complex genomes. Analyzing the sequences of eukaryotes comes with

additional complexity because many genes have alternate ways of being assembled from the linear chromosomal sequence, a feature that is not common in microbes.

A new direction of genome research, known as metagenomics, is focused on understanding the genes produced by communities rather than isolated organisms. Metagenomics is evolving quickly, enabled by continued improvement in sequencing technology. It is also driving new kinds of biological investigations that involve interplay between processes and metabolic pathways that span multiple organisms. This is an important key to understanding any biosystem because humans and microbes generally live as symbionts; for instance, humans are hosts for microbes that are essential to our health.

Regardless of whether one is studying relatively simple microbial genomes, more complex eukaryotes, or extremely complex community systems, the availability of sequence data is both a blessing and a curse. On one hand, a vast and rapidly growing resource of annotated sequence data is available to help characterize newly sequenced systems. But the main drawback is that finding true relationships from complex relationships gets harder as the underlying data set grows. Typical full-genome analysis is already beyond the reach of typical desktop systems unless one is willing to wait days or weeks for the results. Multiple full-genome analysis is even further beyond reach. Unfortunately, this often leads biologists to make an unpleasant decision between disregarding some data in the hopes of bringing the hypothesis to a tractable computing time and accepting sequence analysis as the rate-determining step.

As an example, a recent paper from the Venter Institute [65] details analysis of a large geospatially distributed collection of genome community samples. This analysis required months of computing just to compare all the sequences against one another to feed downstream analysis. For most biologists, this level of computing is not available, but the desire and need to operate at this scale exists. If data are available (which in this case it is) and a hypothesis requires one to operate on that data (which often it does), there should be widely accessible tools for performing the analysis. Similar situations have been faced in astrophysics, chemistry, climate modeling, and a host of other scientific applications. The challenge for DIC is to bring large-scale biological analysis within reach of bench biologists without forcing them to specialize in HPC or algorithm implementation.

3.1.1.2 ScalaBLAST: A Parallel-Processing Algorithm for High-Performance BLAST.

Our solution to the challenges of complexity and data size in bioinformatics are based on DIC principles to (1) solve the throughput challenge with efficiently implemented high-performance algorithms for sequence analysis; (2) solve the time-to-solution challenge by making these tools

available to biologists via easy-to-use interfaces that hide the details of high-performance systems from the end user; and (3) help solve the complexity challenge by presenting intuitive representations of sequence analysis with dynamic, interactive visualizations. In the following sections, we describe how these features of DIC can shift biological sequence research to an environment in which bench biologists have the computing tools and infrastructure needed to generate and evaluate complex hypotheses on massive sequence data sets. This advance shifts the analyst's time from waiting for completion of the computing step to looking at representations of output from computing.

A first step toward creating data-intensive solutions for computational biology and bioinformatics has been the creation of an efficient parallel version of the BLAST algorithm [66,67], which is the workhorse of many bioinformatics analyses. ScalaBLAST [68], originally developed for large-scale BLAST calculations, has been used over the last several years to drive increasingly large analysis tasks using thousands of processors, for instance in the Molecular Sciences Computing Facility supercomputers at PNNL, as well as at other supercomputing facilities. The ScalaBLAST code has allowed users to create very large query and target protein lists for the purpose of finding out which top-k members of the target list are statistically similar to each sequence in the query list. Efficient scaling is achieved by partitioning independent BLAST queries to independent process groups while dividing the database at run-time among processors in a group. This combination of parallel query and database partitioning has resulted in near-perfect scaling to over 1800 processors.

3.1.1.3 Scientific Impact.

The integrated microbial genome (IMG) data resources at the Joint Genome Institute (JGI) have made extensive use of ScalaBLAST to perform BLAST calculations at the scale of hundreds or even thousands of microbial, eukaryal, and viral genomes combined [69,70]. For large-scale calculations used to update the IMG data resources, ScalaBLAST routinely performed BLAST calculations in fewer than 2 days that would take years on dedicated single-CPU machines. The combination of ScalaBLAST to accelerate the underlying BLAST calculations and the IMG data representations and analysis tool suite has proven to be an effective way of putting, indirectly, the power of supercomputing in the hands of bench biologists [71].

The ScalaBLAST code is also used in a different configuration to address the problem of finding how proteins are conserved across species. Many proteins are essential to survival and are common across many species that share some function. As an example, microbes that perform a particular metabolic reaction often have very similar proteins to perform those reactions. Finding how the proteins of a single

organism ‘‘map’’ to closely related organisms is known as ortholog detection. Orthologs are proteins that appear in different species, but essentially perform the same molecular task in each species.

To address multiple-species investigation of orthologs, ScalaBLAST was modified so it retained knowledge of the query species and performed many concurrent (or subsequent) BLAST calculations of a large query list against a collection of separate isolated microbial genomes. This arrangement of ScalaBLAST calculations is known as tiled ScalaBLAST [72]. The resulting alignment data were fed to a separate analysis application (http://en.wikipedia.org/wiki/Inparanoid), which used the BLAST output for a pair of genomes to map the genomes onto each other by identifying which proteins are orthologs. Inparanoid on multiple-species pairs resulted in identification of clusters of proteins from many species that share a task. Driven at a large scale, this analysis is a useful way to transform sequence data from a collection of organisms into a mapping of each genome onto all the others.

A second use of ScalaBLAST for more sophisticated sequence analysis is support vector machine (SVM)-Hustle [73]. In this application, ScalaBLAST scores are used to create a collection of statistical classifiers for recognizing to which protein family a given protein belongs. This is a common analytical technique applied to newly sequenced genes or proteins to gain some understanding of the type of process in which it might be involved before devising wet-lab experiments to verify the function. Protein family prediction with SVM-Hustle requires BLAST scores for all pairs of sequences in the set—a set that numbers in the tens of thousands. While this could have been done with traditional BLAST, the fact that it could be done rapidly with ScalaBLAST allowed SVM-Hustle to be developed and tested in a number of configurations in a short time and will allow it to be retrained rapidly as the public sequence data sets change.

A similar task to protein family prediction is that of protein homology detection. Proteins are homologs when they share similar sequence-related characteristics. Conventional BLAST calculations can identify homologous proteins when they have sequence similarity above 20%, but many naturally occurring protein homolog pairs have sequence similarity below that level. To identify these homologous pairs, we have developed a more sensitive homology tool that uses a statistical learning technique SVM to train with a data set of homologous pairs and nonhomologous pairs. The SVM-based homology tool (SHOT) [74] has been shown to significantly improve the sensitivity of homology detection over that of BLAST. The SHOT must be trained using a large number of homolog pairs (and nonpairs). This training is performed with a vectorized form of the protein pairs that relies on BLAST scores. The ScalaBLAST code is extensively used in SHOT analysis, both in training and in the final application of the SHOT classifier to new sequences.

3.1.1.4 Visualizing Biological Networks. An important difference between DIC and conventional high-throughput computing or HPC is that the patterns in data evident from DIC tend to be much more complex. Having solved the throughput problem for a variety of bioinformatics challenges, it was interesting to see how often the next question is ''now what?'' The fact that ScalaBLAST can perform an all versus all-BLAST calculation of a three million protein data set in less than a day also means it can deliver hundreds of gigabytes (or even terabytes) of output in a day. Eliminating simplifying assumptions in bioinformatics analysis and driving multiple-genome analysis with DIC applications creates a new data problem: How does the end user make sense of all that output?

We have addressed this challenge by linking postprocessing using high-throughput sequence analysis tools (like ScalaBLAST and SHOT) with visual metaphors and visualization applications to create an interface by which bench biologists can devise and test hypotheses.

One example of this is the use of SHOT to associate arbitrary proteins with a basis set of well-characterized proteins to form cursory star clusters. Figure 5 illustrates the use of such star clusters where the central node is a well-characterized basis

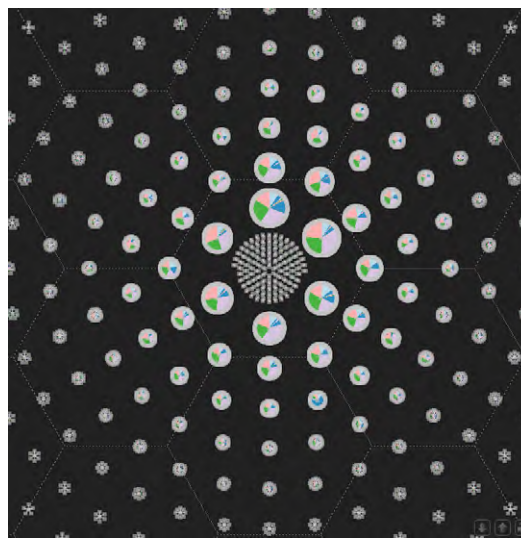

FIG. 5. SHOT output visualized using ''star'' representation. Each central node is a well-characterized SCOP protein, and each peripheral node is a protein from one of five uncharacterized species. Edges are colored to indicate the identity of the species. Large radius stars have large number of related proteins, giving the appearance of pie charts.

protein (in this case, a protein from the SCOP database http://scop.protres.ru/) and the color of the edge conveys the identity of the species having the link. This has been used for multiple-genome analysis for up to 10 closely related species. Working directly with microbiologists has revealed the utility of this representation, which from a distance, allows analysts to determine which basis proteins are over- or underrepresented in a given species.

A second example of the value added to data-intensive bioinformatics by visualization is evident when looking at ortholog clusters across multiple species. Illustrated in Fig. 6 are increasingly fine-grained views of ortholog clusters across 10 *Shewanella* species [74]. In this study, ortholog clusters are used in tandem with SHOT star clusters to identify and refine protein groups that had unusual distribution in the 10 genomes as well as to identify novel functional roles that these proteins had based on their association to basis proteins and each other. This analysis is performed without any prior expectation of a hypothesis of interest and hinged entirely on iterative analysis done on multiple genomes. Ortholog clusters can be viewed with this technique either in tandem with SHOT output or as a straightforward visualization method for multiple-species mappings.

3.1.1.5 Closing the Loop Between High-Performance Computing, Analysis Tools, and Visualization.

Visual metaphors to represent complex patterns in biological data can be a powerful vehicle for making the output of high-throughput bioinformatics accessible to researchers. This can be implemented with a variety of approaches, including some of those already described in this chapter, as well as Web-based front-ends to large-scale compute resources. Web services or applications are commonly used to enable bioinformatics enhanced by specialized hardware transparently to the users. This can be thought of as a straight-through pipeline for high-throughput bioinformatics (Fig. 7).

Closing the loop between visual metaphors and specialized hardware is an even more powerful DIC methodology that is implemented for use in bioinformatics applications. In this computing model, one feeds a visual representation with multiple-genome data analysis output (e.g., from BLAST or SHOT) depicting the relationships between data elements in a cursory view. From this view, biologists can drill down, selecting portions of the data for more analysis and computing, as illustrated in the bottom panel of Fig. 7. Allowing researchers to arbitrarily drill down or up through data, even when doing so requires large-scale computing, creates a capability in which potentially a large number of preliminary hypotheses can be inferred and tested from a single data set. Also, different analytical capabilities can be brought to bear on user-defined subsets of the data where the use of these

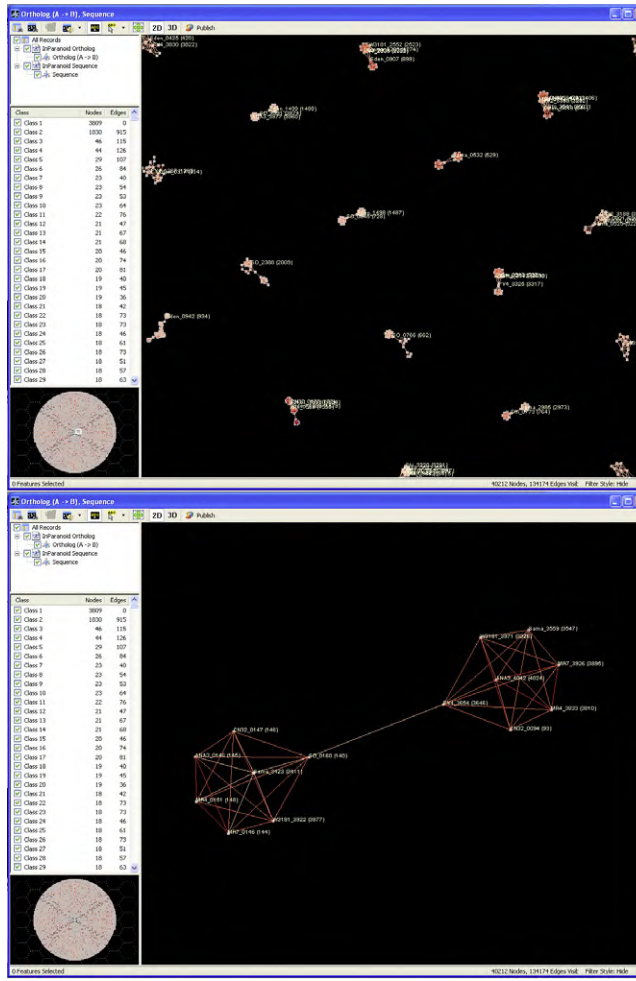

FIG. 6. Top: ortholog clusters for 10 *Shewanella* species. Orthologs can cluster in complex arrangements that are very difficult to search for, but easy to interpret when found. Bottom: one cluster has joined two close ortholog groups, each being represented by seven species. The protein pair linking the two tight clusters may have dual functionality.

methods on the full data set is intractable. The key to this approach is to ensure that, at each level of refinement, enough computer capacity and efficient algorithms are available through the interface to ensure that rounds of refinement and coarsening take a short time. We have previously demonstrated this as one way of allowing the

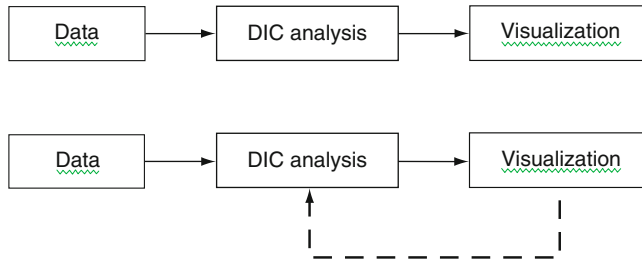

FIG. 7. Top: straight-through analysis allows users to visualize complex patterns in data revealed by DIC. Bottom: allowing users to direct DIC computing tasks based on what patterns they see in the visual metaphors makes the pipeline a more dynamic and interactive tool, enabling real-time hypothesis generation and testing.

data, and the patterns in it, to direct generation of hypotheses, thereby allowing one to find the unexpected [74]. This is potentially a revolutionary approach to biology where researchers do not decide *a priori* what the hypothesis is, but rather it is inferred from the data. This can be a powerful tool to augment more classical hypothesis-directed biological research.

3.1.2 High-Throughput Proteomics Analysis

Proteins are the ultimate functional macromolecules in cellular systems, and the fraction of potentially expressed proteins actually present in a cell, tissue, or biofluid at a given time is strongly dependent on the organism environment and physiological state. In addition, the functions of proteins also may be moderated by posttranslational modifications, such as phosphorylation, or by complexation with other biomolecules or small molecules.

Recent years have seen a rapid increase in the research referred to as proteomics. Generally, proteomics is defined as the study of proteins to obtain global measurements of abundance levels, posttranslational modifications, interactions, and locations. A wide variety of technologies are being used in the field of proteomics. One of the most widely used technologies is known as ‘‘shotgun’’ or ‘‘bottom-up’’ proteomics, in which a complex protein mixture is digested with a protease and analyzed with tandem mass spectrometry (MS/MS) coupled with LC [75–80]. In this approach, digested pieces of the proteins, called peptides, are passed through a cylindrical liquid column. As peptides elute through the liquid column, they are fragmented in a mass spectrometer and the fragmentation spectra are collected. Interpretation of the fragmentation spectra is performed using search algorithms that

match these patterns to theoretical patterns constructed from translation of sequence databases described above. The matches identified by this process are then used to infer the proteins present in the sample. This approach has substantial resolving power to identify a large fraction of the proteins in a complex mixture. This technology is characterized as highly data intensive and relatively low throughput.

Typical proteomics experiments regularly search hundreds of thousands of spectra against databases containing anywhere from a few thousand proteins to hundreds of thousands of sequences. While search times do not scale linearly with increasing database sizes, searching for modifications on sequences increases the complexity of these searches multifold. A single-modification search with InsPecT [81], a popular search engine, takes 1 s per spectrum against a 30 MB database. Multiple levels of complexity are introduced by increasing database size, searching for posttranslational modifications, and complexity of the biological sample under investigation. For example, research in microbial communities typically generates databases with millions of proteins [7] and poses challenging questions to researchers. A common approach to reducing database sizes is the use of a clustering technique based on either sequence composition or functional characteristics. Clustering of very large biological databases is an area of active research, and parallel-processing algorithms are becoming more common [82] because of the data-intensive nature of the problem. Sequence-based clustering of biological data may require the computation of a pairwise similarity measure (Smith-Waterman scores or BLAST scores) between all possible sequences. The ScalaBLAST application described above provides efficient computation of the all versus all-BLAST scores matrix.

To mitigate this limitation in protein characterization, there has been broad interest in the development of approaches that are able to both perform high-throughput analyses and provide protein identifications with reduced experiment times. At PNNL, a method has been developed based upon the use of accurate mass and elution time (AMT) tags [76,83,84] in which a database of identifications is developed from the result of the time-consuming MS/MS experiments described above. The elution time is measured from the LC column while the mass spectrometer outputs a mass/charge measurement that is deconvolved to obtain an accurate mass measurement. Current generation mass spectrometers exhibit very high sensitivities and increasing dynamic ranges to detect peptides accurately within a few parts per million (ppm). Patterns of mass and elution time in subsequent LC–MS/MS experiments (in which fragmentation is not performed) are matched to this curated database to infer the presence of peptides in the sample, which are eventually rolled up to identify the proteins.

Figure 8 graphically illustrates the AMT tag approach. A significant challenge in identification with the AMT approach is eliminating the variation in elution time

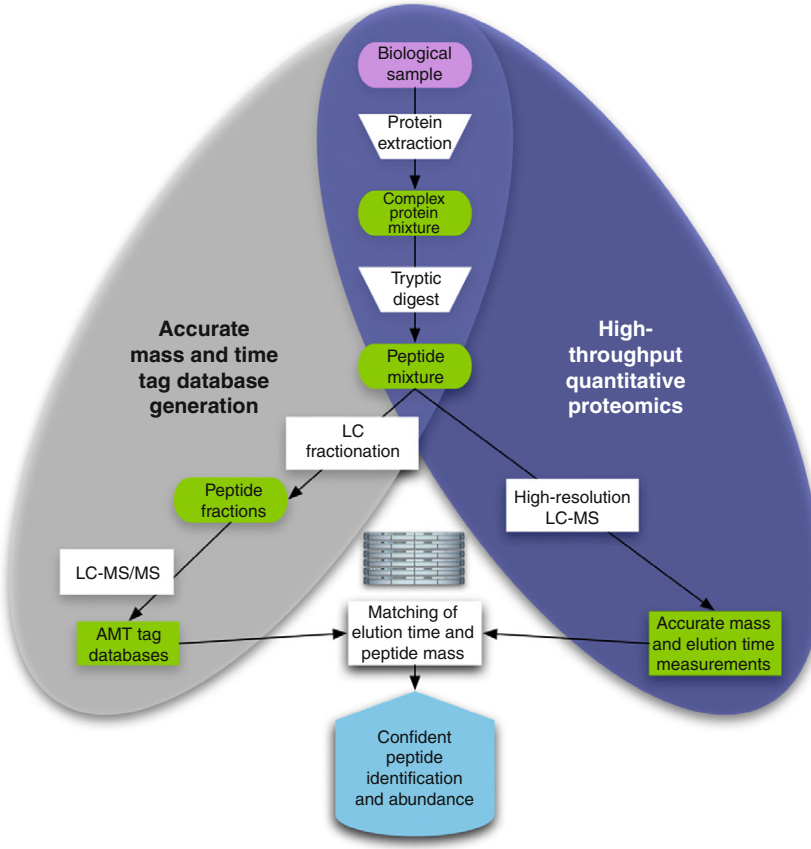

FIG. 8. Accurate mass and time (AMT) tag proteomics approach.

across experiments. One solution to this problem is to use a predictive capability to calculate elution times based on a peptide sequence [85]. Another approach focuses on aligning data sets across multiple experiments against a common baseline and against an existing AMT tag database [86]. Subsequent experiments with the AMT approach do not spend any time fragmenting peptide signatures and are relatively high throughput. Current throughput at PNNL allows about 10 global proteome measurements per day, per instrument. This throughput rate can be compared to the days or weeks required for similar levels of coverage required for a single traditional ''shotgun-derived'' global proteome analysis.

3.1.2.1 The Tandem Mass Spectrometry Challenge.

Perhaps the most significant computational challenge in proteomics is the identification of peptides from MS/MS data. High-throughput technologies such as the AMT pipeline still depend on the development of databases from MS/MS analyses. These MS/MS experiments are performed independently and sampling of ions is a probabilistic process, which can be modeled as a logistic distribution with more abundant peptides and provide near-complete detection of proteins present in a biological study. Additionally, analysis of redundantly collected spectra adds to computational analysis time without adding to the information generated. Finally, the added complexity from posttranslationally modified peptides or unexpected single amino acid substitutions results in exponentially increasing computational needs.

In comparison to proteomic analysis of single organisms from pure culture, analogous proteomic assessments of microbial community samples are difficult at best, and often involve undesirable assumptions about the data to make the analyses computationally feasible and less error prone. For example, the identification of nontryptic peptides or more than one type of posttranslationally modified peptide from microbial community samples increases the number of candidate peptides to a level at which the computations do not finish in a reasonable time with the use of current codes and computer architectures (Fig. 9). Furthermore, for most microbial communities, sequence information is not available, meaning that a database of proteins (and hence peptide candidates) for a microbial community is an incomplete estimate, at best.

To illustrate the increasing difficulty of identifying peptides with increasing sample complexity, consider that each MS/MS spectrum must be assessed in the context of a database of candidate peptides, and that the databases grow by orders of magnitude with the complexity of the sample source, as shown in Fig. 9. In addition, expanding the search to include a single type of posttranslational modification further increases the search space by another order of magnitude. For an MS/MS spectrum from a microbial community sample, including four common types of posttranslational modifications means that the detection method must select the one correct candidate out of nearly 10^{11} possible candidates. This leads to results containing either an unreasonably high number of false identifications or a large number of spectra being discarded.

While several groups have developed methods for making assumptions about the data to reduce the time to solution [87,88], much less work has been done regarding the mathematical models used in the identification process to improve accuracy or the examination of different computer architectures for processing the data.

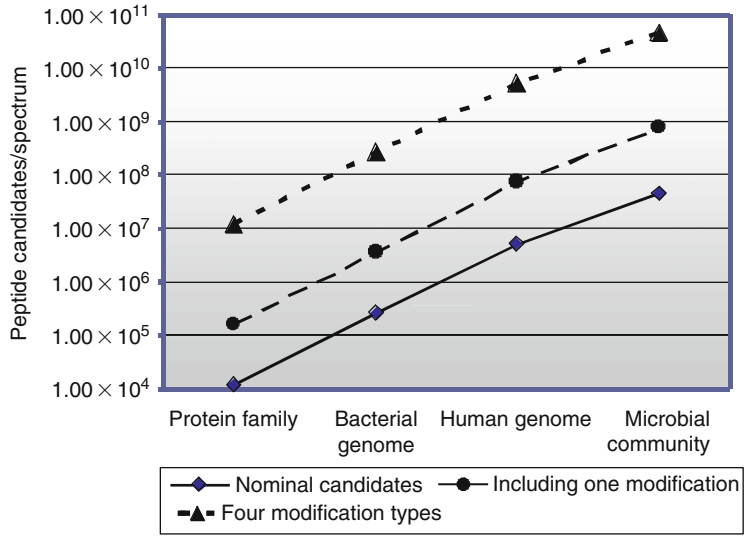

FIG. 9. Number of tryptic peptide candidates for each individual MS/MS spectrum grows by several orders of magnitude when posttranslational modifications are considered. The term ''Nominal Candidates'' refers to peptide candidates containing unmodified amino acids. Also shown is the number of candidates to be considered when one or four types of posttranslational modifications are allowed.

The development of advanced model spectra from spectra libraries [89–91] for detecting highly specific fingerprints of the peptide spectra results in increased accuracy and a greater number of spectra being identified. One of the challenges has been to integrate the use of model spectra from spectral libraries with previous identification methods that rely on *ad hoc* model spectra and *ad hoc* measures of similarity between model spectra and experimentally observed spectra. Our approach to this challenge has been to develop both our statistical similarity measure and our model spectra on the principles of statistical mechanics that govern the physical and chemical processes in the laboratory. The end result is a single analysis method that can use either model spectra derived from either spectral libraries or averaged training data on diverse peptides.

As shown in Fig. 10, such an approach can at least double the number of peptides that are identified and will much more precisely elucidate the proteomic state of a cell population by identifying a larger set of proteins that are expressed under specific growth conditions. The *x*-axis shows the percent increase in the identification rate, while the *y*-axis shows the number of peptides at that increased level of identification. On average, the 44 peptides were identified with 2.4 times more

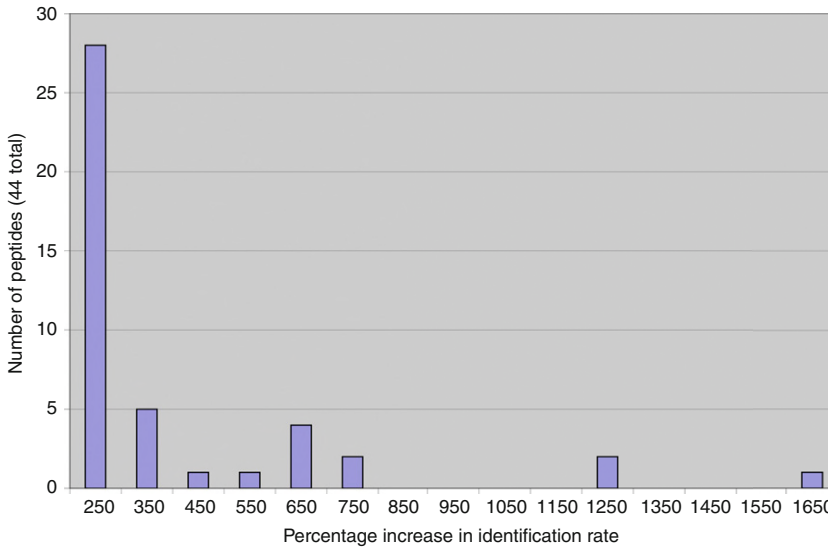

FIG. 10. Histogram of the percent increase in the number of spectra identified for the 44 peptides for which spectral libraries were available.

spectra when realistic model spectra derived from experimental data are used rather than generic, one-size-fits-all approach to generating model spectra.

Although the use of spectral libraries may be the most significant advance in peptide identification since 1994, it presents a significant challenge with respect to data management and throughput. One consequence of spectral libraries is that a model spectrum for each peptide needs to be stored in a database and that database needs to be readily accessible for use in HPC programs. This challenge becomes particularly significant when one considers that peptides that have posttranslational modifications have spectra that are different from the unmodified form of the peptide. As shown in Fig. 10, if complete spectral libraries were available as a result of either experimentation or computation, the database would be on the order of several exabytes. We have been addressing this challenge by developing code that allows for fault-tolerant access to global storage resources, both memory and disk. The code has been implemented and tested on architectures that allow for globally accessible memory (SGI Altix) and on large commodity clusters that use global file systems (Chinook, an HPC Linux cluster at the Molecular Sciences Computing Facility at PNNL with 2310 nodes/4620 quad-core processors on which the Lustre file system is used).

Furthermore, as these methods progress, it will be desirable to reanalyze existing data sets with the new methods to elucidate the understanding of the biological processes involved in the samples. However, the computational reanalysis of existing data sets of MS/MS spectra will be extremely demanding. For example, the Environmental Molecular Sciences Laboratory at PNNL currently stores over 21 million spectra for *Shewanella oneidensis MR-1* alone, and over one billion spectra all together. The reanalysis of the data will be time-prohibitive if current software and hardware capabilities are used.

3.1.2.2 Increasing Throughput Through Smart Instrument Control. Our proposed high-performance, data-intensive analysis pipeline (Fig. 11) builds on the key components used in the current AMT tag process to convert the current static process into a novel, dynamic approach that brings the *in silico* analytical process to the instrument rather than decoupling the two. This capability was not possible previously because of our inability to do online processing of the captured data. Using MeDICi as our underlying architecture, we overcome the above limitations and provide enhanced information extraction that automatically analyzes incoming data and makes intelligent decisions, based on

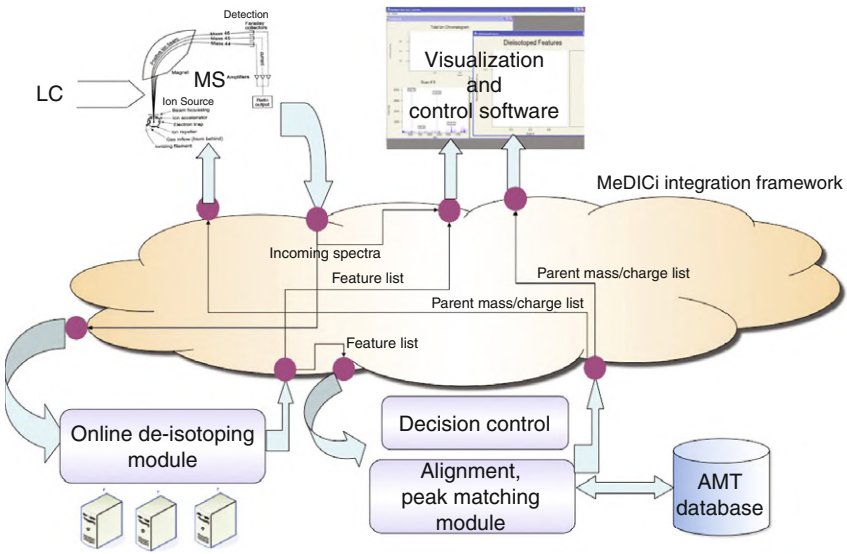

FIG. 11. High-performance data analysis pipeline for smart instrument control.

statistical decision theory, to sample new ions intelligently and iteratively augment data collected. This technology allows experiments to move into the paradigm of true large-scale experiments rather than a large collection of independent experiments. Complex biological samples are characterized in more depth and at a faster rate, thus facilitating systems biology. In addition, exploration of unexpected events also could be feasible through the possibility of controlling the experiments dynamically.

This system would also make feasible the exploration of new paradigms, such as one in which multiple mass spectrometers may be intelligently controlled in a coordinated manner with a scalable, online computational system to perform biological analyses previously not imagined.

The smart instrument control technology builds upon several existing components that have been developed to process mass spectrometry data in a retrospective manner. The individual components are:

- *Feature discovery algorithms*. Data generated by a mass spectrometer coupled to a separations system, such as liquid chromatography, which adds a time dimension to an experiment, includes a series of mass spectra acquired over time. Each ion in an analysis is observed in the mass and time dimensions with an intensity pattern that is somewhat predictable from its chemical formula, its abundance, the charge it carries, and the resolution of the instrument. However, during the course of an analysis, the actual formula is unknown and discovering features is further complicated by the presence of noise and overlapping ion signatures. We have developed and implemented internally and externally developed algorithms [92–95] to discover features in an individual mass spectrum in an unsupervised manner and to group together the same features in consecutive spectra [96]. These algorithms are currently available to the proteomics community in two software packages: Decon2LS and VIPER [96] at http://ncrr.pnl.gov/software. Our initial experiments suggest that the feature discovery process is slower on spectra containing much information. The advantage of using MeDICi as our underlying pipeline allows us the flexibility to connect HPC hardware such as the FPGA to deisotope spectra. The feasibility of using FPGA for fast deconvolution of mass spectra has been illustrated by Bogdan et al. [97].
- *Alignment algorithms*. Features discovered in mass spectrometry data are identified by matching them to a database of previously identified peptides on the basis of mass and time values. However, before matching can be performed, corrections in variability in mass and elution time dimensions need to be made. We have developed a dynamic-programming, time-warping algorithm which finds the transformation function in the mass and time

dimensions and results in the maximum likelihood of the data being observed from the database [86].

- *Database searching algorithm.* Fragmentation spectra collected during an MS/MS experiment need to be interpreted using a search algorithm [81,88,98,99] that matches them against theoretical patterns of peptides in a species being analyzed.
- *High-performance DIC using MeDICi.* The most important piece for the implementation of this technology is the use of a robust underlying plumbing mechanism that supports data distribution, multiple types of hardware and software modules, and reliable data delivery. Our DIC architecture, the MeDICi integration framework, in addition to simplifying the pipeline creating process also facilitates the data handling and transformation of data while passing through the pipeline to maximize the performance of the applications. The architecture also enables analytical codes written in any language and running on any platform to be plugged into a MeDICi pipeline, simply through the creation of a few lines of code, and without changing the analysis code itself.

Our first-generation version of this pipeline implements a serialized framework in which events are collected synchronously. Mass spectroscopy spectra are directly pumped into the MeDICi integration framework that is responsible for reliable, in-order delivery to a number of software processing modules, the feature discovery module, and a central visualization and control software. The feature discovery module interprets the mass spectrum with our current deisotoping algorithm, which is a refinement of the THRASH algorithm [92]. This feature list is then forwarded to an online alignment and peak-matching module. This module compares the features to those existing in a database and sends appropriate feedback to the mass spectrometer with a list of parent ions to be fragmented in the next fragmentation cycle. This avoids the fragmentation of previously observed species and helps the mass spectrometer delve into the lower level features on a spectrum-by-spectrum basis.

3.1.2.3 Scientific Impact.

Mass spectrometry promises to be a valuable medium in discovering biomarkers for diseases that may have multiple underlying causes, such as cancers [100–108]. There has been significant interest in the use of mass spectrometry to discover biomarkers; one modern and well-publicized example was published by Petricoin et al. [109]. This particular example used mass spectrometry-based patterns to classify patient blood plasma samples as coming from women with ovarian cancer or noncancer. The research has been considered

controversial for multiple reasons. One of the major reasons has been the classification based on highly abundant low-resolution mass spectral peaks without any reasonable means to perform specific identification of the measured species and patterns [110]. The other major concern was that the number of samples used in the study resulted in low statistical significance. Further, the specificity of the measurements resulted in unacceptable numbers of false-positive cancer detections, leading to worry for the patient and substantial follow-up costs for the health care system [111,112]. This study and the associated controversy highlighted the need for ''smart instrument control'' methods in which (1) the identity of each analyte was considered important and possibly relevant in biomarker discovery, (2) methods were needed to mitigate the effects of highly abundant species which were generally not considered useful even for classification purposes, and (3) the methods used were capable of at least hundreds of individual measurements with extremely high sensitivity and specificity.

Smart instrument control methodologies allow measurements to be performed that depend on the results of a previous measurement or set of measurements. This allows sophisticated strategies that, for example, minimize the remeasurement of previously identified species when building a database of expressed peptides. In conventional experiments, many repeated experimental and sample fractionation methods are performed to achieve the needed detection of a broad dynamic range of peptides. These advanced instrument control methods require the ability of move large volumes of data and process the data streams in real time.

3.1.3 Inverse Hadamard Transform for Ion Mobility Separation Coupled with Mass Spectrometry

Mass spectrometers measure the mass-to-charge ratio of ions present in a physical sample with a high degree of accuracy and precision. The analysis of complex mixtures, such as those found in proteomics analysis, with mass spectrometry benefits from the coupling of separation technologies, such as liquid chromatography with mass spectrometry. This separation spreads the complexity across hundreds to thousands of spectra. As the sample complexity increases and throughput requirements continue to increase, the need for more sophisticated strategies are justified. At PNNL, we have added ion mobility separation (IMS) between the liquid chromatography and mass spectrometry systems, thus allowing ultrahigh-throughput analysis of complex mixtures. In IMS, ions are moved through a background gas in a drift tube. An electrical field is applied to the drift tube, thus forcing the ions to drift through the tube. The ions drift at a speed based on their cross-sectional area, with

more compact ions reaching the end of the drift tube first. A time-of-flight (TOF) mass spectrometer is placed at the end of the drift tube to analyze the ions present as a function of drift time. While unique information regarding an ion's size may be determined with IMS, the sensitivity and overall utility of the technique has been limited because of its inherently low duty cycle. This is because a packet of ions must clear the drift tube before the next batch can enter. Use of multiplexing techniques, such as the Hadamard transform [113], may have significantly increased the utilization. The captured data have been physically convolved by the Hadamard encoding process and must be computationally deconvolved to obtain the actual sample data. However, applying the Hadamard transform to increase the instrument utilization requires real-time online processing to deconvolve the data to support smart instrument control methods.

3.1.3.1 Hadamard Transform.

The encoding random sequence used to control the ion gate at the entrance to the drift tube can be represented as a Simplex matrix. Simplex matrices are composed of ones and zeroes and have dimensions $N \times N$, where $N = 2^m - 1$, $m \in N$. The number of ones and zeroes in any particular row is approximately equal, thus accounting for the 50% utilization achieved by mass spectrometers with this technique.

The dimensional size (N) or *order* of the transform is chosen based on the range of flight times being used in the instrument. All Simplex matrices are either *right circulant* or *left circulant*, which means that the elements of each row of the matrix are equivalent to the elements of the previous row of the matrix shifted by one column to the right or left, respectively. The physical convolution of the data is equivalent to

$$y = S_N x + e,$$

where y is the data vector observed at capture time, S_N is the selected Simplex matrix of dimensional size N, x is the spectrum data vector that has been obfuscated by the physical convolution, and e is a time-independent noise vector. More details on Simplex matrices and their use in the Hadamard encoding process can be found in the literature [114].

To perform the computational deconvolution of the captured data (y), the inverse of the Simplex matrix (S) used in the encoding must be utilized. The captured data vector is multiplied by the inverse Simplex matrix (S_N^{-1}) to obtain the expected set of spectra (each spectrum is referred to as scans, i.e., the frame) that represents one IMS. Each spectrum or scan is acquired from the TOF with a high-speed analog-to-digital converter to read the ion signal from the detector.

3.1.3.2 Real-Time Processing with Hybrid High-Performance Computing Hardware Architectures. We have implemented the design for the inverse Hadamard transform computation on a single node of a Cray XD-1. The full implementation of the design uses 22% of the slices (5201 out of 23,616) of the Virtex II-Pro XC2VP50 FPGA on the XD-1's node. It uses 36% of the BlockRAM elements (85 out of 232) and only 1% of the 18×18 multipliers (4 out of 232). The design also uses two of the four 4-MB QDR SRAM memory banks available on the XD-1 node. The design achieved a clock frequency of 142 MHz out of a maximum achievable frequency of 199 MHz. The maximum data transfer bandwidth on the XD-1 node is 1.422 GBps. However, this bandwidth can only be achieved under a design that operates at the maximum frequency of 199 MHz. Designs operating at lower clock frequencies achieve a proportional bandwidth. In our case, we achieved a data rate of 1.01 GBps.

Our data set consists of 100 frames of 620 scans each, with 3380 16-bit elements per scan. The total size of the data set is thus 400 MB. The streaming time at approximately 900 MBps is 0.46 s, which includes the ongoing frame accumulation processing. The inverse transform process for the accumulated frame takes 0.11 s and is twice as fast as executing the inverse transform algorithm on the host 2.4-GHz AMD Opteron CPU. The data write-back time for the result of the inverse transform is 4 ms.

Our research has determined that it is feasible to use FPGAs for processing mass spectrometry data at very high rates (GBps range). We have also shown that is possible to implement domain-specific signal processing algorithms (inverse Hadamard transform) to operate in conjunction with high-speed data capture.

3.2 Data-Intensive Cyber Analytics

Ensuring the security and reliability of communications systems requires the ability to process vast amounts of transactional records for indicators of problems, often in real time. Such systems include computer networks, voice networks, and power-grid control systems.

Cyber security poses particularly acute analytic challenges, not least because of the immense volume and complexity of the activity that occurs over computer networks. A typical enterprise computing network might generate billions of packet transactions per day, each a subcomponent of a larger activity such as Web surfing, but some that might also be indicative of more malevolent activity. Moreover, unlike domains where the endpoints in a transaction are well defined, such as could be the case in financial transactions or communication between power-grid components, there is a tremendous variety of activity occurring over computer networks.

Generally, while the source and destination IP addresses involved in a transaction are known, the exact applications involved in the transaction are not. Finally, the number of unique ''actors'' on a computer network can be large. Specifically, the number of machines on a network being protected might number in the hundreds or thousands, and the number of external IP addresses to which these machines might connect is several orders of magnitude larger, for example, there are 2^{32} unique addresses in IPv4 and 2^{128} in IPv6. The number of new malicious code threats also has increased dramatically, growing from 140,690 in 2006 to 1,656,227 in 2008 [115].

While the sensor technology to instrument and transmit records of each transaction in a computer network exists, analytic techniques to detect and characterize security events are less well developed. These events are sometimes ''anomalous,'' that is, they are statistically separable from ''background'' communication, but not all anomalies are malicious nor are all malicious activities easily recognized as anomalies. Indeed, the more sophisticated an attack, the more the attacker will attempt to make his behavior appear normal. Many approaches to detecting computer security problems take a deterministic approach, in which signatures of known malicious activity are used as patterns against which to match current traffic. While these methods are generally effective at detecting instances of previously discovered threats, they are not well suited to discovering new threats quickly or to enabling timely triage, decision making, and response actions by analysts.

3.2.1 The Cyber-Security Analytics Challenge

The cyber analytics challenge, therefore, is to enable the full range of ''human-in-the-loop'' discovery and detection such that both high-level situational awareness of broad patterns in data and detailed analysis of potential threats involving small amounts of signal in an overwhelming amount of noise are possible. While entirely automated approaches to security are desirable, the reality of sophisticated attacks means that humans are almost always needed to discover, or at least confirm, potentially malicious activity. Cyber analytics research must focus on how best to support human cognitive abilities; when dealing with high-volume streaming network traffic, this means leveraging the innate perceptual abilities of humans to detect off-normal conditions without inducing cognitive overload. Our approach to data-intensive cyber analytics involves two complementary processes: data reduction and visualization.

Data reduction can involve the development of models, signatures, or statistics, all of which can help aggregate large amounts of low-level data into more information-dense representations. The aim of these derivative representations is to preserve

the structure of the underlying data with as much fidelity as possible, while reducing the volume of what must be analyzed to a level more appropriate for human inquiry.

Visualization involves the depiction of structure, trends, or relationships in data such that humans can maintain awareness of system state and can explore, form, and test hypotheses. Neither data-reduction techniques, such as modeling, nor visualization is typically sufficient when used in isolation; however, when the two processes are used in tandem, new mixed-initiative systems that support automated detection and human-driven exploration are possible. The following sections introduce a selection of data-intensive cyber analytics applications and outlines future needs for continued development.

In cyber analytics, a central goal is to understand the processes by which actors are engaged in potentially malicious activity. Cyber analytics typically (although not always exclusively) proceeds in a top-down fashion; antivirus tools or intrusion detection systems often use signatures of known events to classify new events. Malicious actors also have a limited and familiar set of motivations for attacks, such as fame, fortune, power, or revenge. These high-level motivations may be fulfilled in a variety of common attack vectors, such as denial of service, eavesdropping, or man-in-the-middle activities, in which attackers intercept and modify communications between other parties. But because attack vectors can vary in the details of their implementation, focusing on low-level ''bits on the wire'' can make it difficult to recognize and respond to problems. Thus, cyber analysts often search for events that match high-level descriptions of known problems, and then proceed by interpreting observed events from the top down. Context, such as knowledge of current world events and computer security vulnerabilities, helps analysts focus their searches. Once a sequence of events or data is recognized as being constant with respect to the appearance of higher level symptoms, it can be used as a signature for rapid future detection.

In the cyber-security field, patterns are often referred to as attack signatures. Usually, these signatures are specific to a particular attack tool or method and have a single data source. While there is considerable existing work on the identification of individually anomalous events in data streams (e.g., [116,117]), an outstanding challenge (and one we address in the following section) is identifying sequences of linked behaviors over time. Prior work has also addressed attack graph visualization [118], but there is a recognized need to develop approaches that scale to operational data volumes, support exploration across levels of abstraction, and allow user-driven pathway definition [119]. Zhuge and Shi [120] draw parallels between the information analysis strategies for computer and biological epidemics, suggesting that the development of cross-domain analytical techniques is an area ripe for breakthrough.

Most contemporary visualization approaches for computer network analysis problems typically focus on node connectivity and traffic flow (e.g., [121,122]) or on firewall alert and packet-level visualization (e.g., [123]). Kafadar and Wegman [124], for instance, characterize ''exotic'' traffic simply as that with particularly high IP address or port frequencies, even though threats may also be carried in seemingly normal or noisy traffic. They do, however, rightly suggest that many existing visualization models are limited to static data sets of moderate size, not the continuously evolving and high-volume data typical of the network security domain. Wegman and Marchette [125] call for new ''evolutionary graphics'' capable of better communicating change over time. Such graphics have broad applicability because the need to succinctly communicate normal and abnormal events in data and to help analysts identify patterns at a range of temporal scales transcends domains.

Visual analytics offers tools to support this cross-scale analysis. Our approach is to aggregate low-level log data to broader behaviors that characterize an actor (malicious or otherwise) over time, while allowing the end user to drill down to more detailed records when needed. This technique reduces, in part, the data overload problem by increasing the level of abstraction at which massive data can be analyzed.

3.2.2 Data-Intensive Analysis Tools

There are two broad categories of analytic work in cyber security: tactical and strategic. Tactical work is concerned with the largely reactive tasks of detecting suspicious activity and assessing its impact on an organization; whereas, strategic work is concerned with identifying larger patterns and motives, and conducting proactive assessments of future attack methods [126]. Additionally, tactical work is typically intended to occur in as near real time as possible, while strategic work may involve less time-critical decision making. While large data volumes are a challenge in both realms, the techniques we describe in this section are designed largely to support the real-time analysis problem. In strategic or forensic activities, often the solution to an analysis problem is the construction of a database to hold records of network transactions for later inspection. Real-time analysis, however, demands new information-management architectures, data-reduction techniques, and human–computer interfaces.

The cyber analysis tools we describe in this section all use network flow data as their primary source of transactional information. While there are many methods for generating them, ''flows'' are essentially sessions that record the attributes of a series of packets between two parties; all of the packets within a particular time window to and from particular ports are aggregated into a single flow. Flow records do not contain the actual content of the packets they aggregate and, although packet

content can be useful in assessing the nature of a potential threat, it is unavailable for many real-world analysis challenges because of encryption or privacy restrictions.

In the following sections, we illustrate complementary techniques for data-intensive analysis. Each application takes a different approach to addressing the problem of data scale, but each is designed to support the human analyst in detecting and understanding events in high-volume data streams. Some applications rely on the use of DIC hardware, while others are handled mostly in software.

3.2.2.1 *StatsView.*

A PNNL developed tool, StatsView alerts analysts to significant changes in aggregate trends of network traffic. It uses three statistical models, each with different assumptions about the behavior of the network, to determine when a statistically significant change occurs. These changes are flagged for the analyst as events that should be investigated thoroughly (Fig. 12). Essentially, StatsView is a triage tool; it examines the entire volume of flow traffic from a sensor and finds cases where current traffic volume in predefined categories departs from the volume seen over the past hour. It provides the most general level of situational awareness, helping an analyst determine whether current conditions are within normally observed boundaries. StatsView, like the other tools described below, is a Java application with the MeDICi architecture to calculate statistics

FIG. 12. StatsView computes several statistics across the full volume of network flows and flags anomalous events for the analyst.

and populate a visualization client with analytic results. The network flows that these tools analyze are collected with sensors at a network gateway and summarize the traffic between an enterprise network and the rest of the Internet. Typically, these flows are bidirectional in that they represent both the ‘‘request’’ and ‘‘response’’ components of a transaction. StatsView incorporates a MeDICi listener that receives streaming flow data from an aggregator. The computation of StatsView statistics could be distributed across multiple analysis nodes, using MeDICi to transmit data to them and deliver processed results to the StatsView client.

3.2.2.2 NUANCE. While tools like StatsView are effective for detecting changes in the aggregate characteristics of high-volume data, some cyber threats may not stand out from the noise of everyday network activity to raise an alert with standard anomaly detection techniques. For instance, an SQL injection attack might be manifested as just a handful of flows; buried in a stream of millions of flows, what techniques can help these few flows stand out? New methods are needed for more finely segmenting traffic to improve the signal-to-noise ratio.

The NUANCE code [127] is an attempt to create behavioral models of each machine on a network, so anomalies at the level of an individual host, rather than an entire sensor, which may aggregate data from multiple hosts, can be detected. While there are existing host-based intrusion detection and activity profiling systems, NUANCE uses a modeling technique that characterizes the expected activity of each machine on a network over multiple time periods from minutes, to hours, to days. These models provide analysts with a unique view into the baseline behavior of their hosts. This is a level of insight that many argue is crucial to improving situational awareness (Fig. 13). The current network activity to or from a host is compared to the model results and departures from expected activity above a user-defined threshold could be flagged for the analyst.

The NUANCE behavior model consists of parsing, statistics, and curve-fitting components, connected through MeDICi. The parser receives real-time network data and summarizes each record as a timestamp and a list of ‘‘groups’’ to which the transaction belongs. Group membership is assigned by rules such as ‘‘Is this packet from a .edu domain?’’ ‘‘Is this an HTTP packet?’’ or ‘‘Is this an HTTP Packet from ABC Co.?’’ The statistics thread monitors output from the parser and maintains an array of sufficient statistics for each IP address and group. For a given actor, the statistical model expresses the expected traffic rate over time as a periodic function with an exponentiated Fourier series.

Techniques, such as NUANCE, help address the data-intensive cyber analytics challenge through data reduction. By segmenting traffic by actor and then looking only for actors who appear to be behaving anomalously, NUANCE helps target

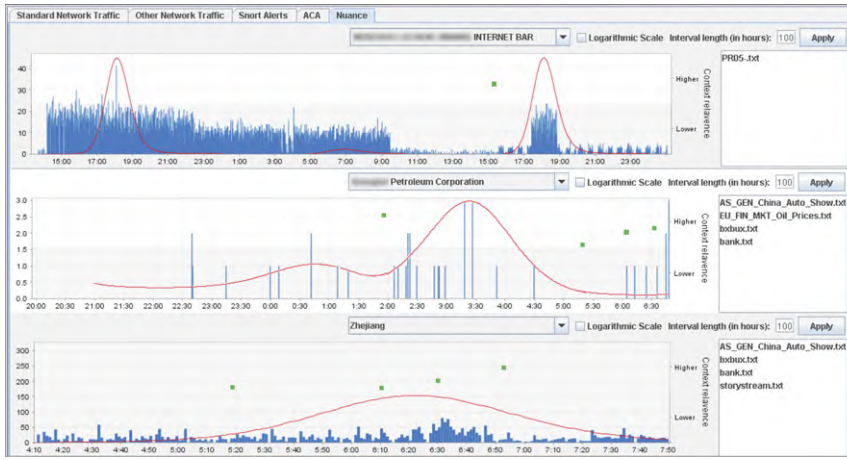

FIG. 13. NUANCE crafts a behavioral model (red curves) for each actor on a network, which it compares to observed activity (blue histogram) to identify smaller anomalies than aggregate anomaly detection techniques would capture.

human attention to only those portions of data most worthy of scrutiny. It also introduces visualization into the analysis workflow at a later point than traditional discovery tools; rather than display the full volume of network data for the analyst to explore, visualization is used only after specific events are detected in that data. The NUANCE code has been used to model traffic for test cases on the order of hundreds of network actors on a single processing node. The ability of MeDICi to distribute the modeling load allows NUANCE to scale readily by adding additional modeling nodes.

3.2.2.3 CLIQUE. While looking for anomalous behaviors in NUANCE provides insight into streaming data that is otherwise hard to obtain, there is still the challenge of discovering sets of related behaviors over time. For instance, a cyber exploit, such as a rootkit that surreptitiously exfiltrates data from a user's machine, might have precursor phases whereby the attacker first performs reconnaissance on target machines, attempts to install the malware, etc. Rather than detect the anomalous transmission of large amounts of data to an external site, which is statistically comparatively easy to detect, but by which point the damage may already be done, it is desirable to understand how the components of an attack behavior relate to each other such that precursor events can be detected and presented to the analyst for action.

The CLIQUE code is based on the premise of ''correlation layers,'' or components for iteratively abstracting data into correlated sets. In the cyber-security domain, network packets first might be correlated or aggregated into sessions. These sessions, in turn, can be correlated into longer term, evolving behaviors for the parties in that session and these behaviors then correlated into linked collections of activities that represent activities associated in a common purpose—a ''clique.'' The NUANCE models cannot make associations between events over time to identify larger scale cyber activities. To overcome this limitation, a real-time streaming classifier is used in CLIQUE to group network flows into bins of like activity. The categorical classifier finds natural groupings among flows. Once each flow is tagged with a category, a sequence identification process examines the pattern of categories that each IP address manifests. Through this approach, it is possible to detect and permit sequences that are typically benign, while highlighting for the analyst sequences that are typically only seen in relation to known malicious activity or seen rarely.

3.2.2.4 Partial Dimensions Tree. The Partial Dimensions Tree (PDTree) application involves large sets of network traffic data [128]. Analysis is performed to detect anomalies in network traffic packet headers to locate and characterize network attacks and to help predict and mitigate future attacks. This application is a special case of a more widely applicable analysis method to find relationships and patterns in the data [42]. It uses ideas from conditional probability in conjunction with a novel data structure and algorithm. When dealing with multivariate categorical data we can ask, for any combination of variables and instantiation of values for those variables, how many times this pattern has occurred? Because multiple variables are being considered simultaneously, the resulting count or contingency table specifies a joint distribution. The massive data volume prevalent in cyber-security analyses has a large number of variables and observations, as well as many distinct values for variables. A PDTree data structure is a practical simplification of the All Dimensions Tree (ADTree) data structure described by Moore and Lee [129] that can store the instance counts for different combinations of variables. The PDTree structure requires less memory than the ADTree when used in conjunction with domain knowledge about the expected variable combinations. On the Cray XMT, the PDTree is a multiple type, recursive tree structure. Figure 14 illustrates the absolute performance and relative increases in computational speed obtained with the PDTree code analyzing a one million record and a 500,000 record data set derived from cyber-security traffic data on the Cray MTA-2.

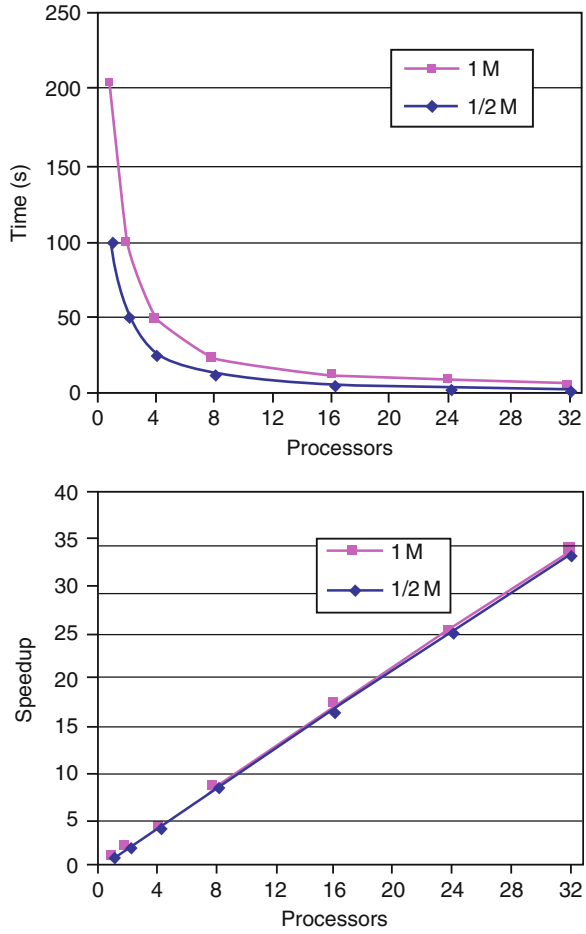

FIG. 14. Wall clock time (left) and speedup (right) for 1/2 and 1 million records on the Cray MTA-2 [128].

We have extended our original PDTree implementation to handle very large data sets with the Cray XMT hybrid execution capabilities through a Lightweight User Communication (LUC)-based remote procedure call (RPC) mechanism. We have developed a LUC server to execute on the Opteron service nodes, which have direct access to a large Lustre file system. The server opens a very large file with PDTree data resident on the Lustre file system. The server then proceeds to stream chunks of

the file in response to requests from a Threadstorm LUC client. The server reads the file in binary mode in fixed size chunks and sends the data to the Threadstorm client after discarding incomplete records that might appear at the end of the read chunk. Each one of the read chunks is then processed as a record set and inserted into its corresponding place in the PDTree. This process closely follows the expected use of a PDTree-like application for anomaly detection in a realistic networking environment with streaming data coming from network routers and sensors. Figure 15 illustrates the streamed execution style. A more detailed discussion is provided by Chavarria-Miranda et al. [41,128].

3.2.2.5 Future Needs. While current data-intensive cyber analysis tools are beginning to change the dominant analytic paradigm from ''data-object''-level analysis, that is, displaying individual flows for the analyst, to derivative ''feature''-level analyses, that is, displaying derived behaviors or patterns that aggregate low-level flows into more abstract representations, there remain significant outstanding challenges. First among these challenges is better feedback between human and automated discovery processes. To create effective mixed-initiative systems, it is necessary for automated approaches to be trained by their users and to evolve over time based on their users needs. For instance,

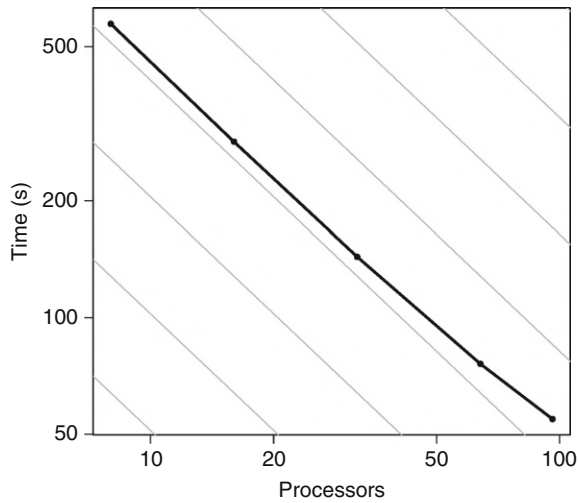

FIG. 15. Wall clock time (left) and speedup (right) for PDTree streamed execution on a preproduction Cray XMT using five chunks of ~4,100,000 records from a 4-GB Lustre [128].

analysts need to be able to provide feedback like ''yes, show me more of these kinds of events'' or ''no, the anomaly the system has flagged is benign, don't show me another case of it.'' In turn, automated systems need to be able to challenge the user hypotheses, perhaps by playing a ''devil's advocate'' role and suggesting alternative explanations for events. This degree of cooperation is necessary to ensure that large data spaces are adequately assessed. Second, substantial new research into the most effective visual representations for high-volume data is needed. Typical cognition and perception studies involve small sample data sets, but methods for supporting human understanding of what is going on in a data space of hundreds of millions of features or more are underdeveloped. What are the best ways to summarize vast amounts of network traffic to users such that important patterns or trends are highlighted, but potentially important features are not ''averaged out?'' Scalable visualizations—interfaces that simultaneously support focus and context—are necessary. Third, analytic principles need to be generalized across domains so common tools and techniques can be applied to the wide of application areas challenged with high-volume transactional data from computer networks to supervisory control and data acquisition (SCADA) systems to financial fraud.

3.3 Applications in Atmospheric Radiation Measurement Program

The Atmospheric Radiation Measurement (ARM) program [130], a multilaboratory, multiorganization program, is the largest Climate Research program supported by US DOE. The ARM Climate Research Facility (ACRF) is a DOE user facility responsible for design and maintenance of the ARM Data System Infrastructure. In 1989, DOE's Office of Science created the ARM program to address scientific uncertainties related to global climate change, with a specific focus on the crucial role of clouds and their influence on the transfer of radiation in the atmosphere. Heavily instrumented sites have been established at three primary locations (Fig. 16): (1) the Southern Great Plains in United States, (2) the Tropical Western Pacific, and (3) the North Slope of Alaska. Additionally, two mobile facilities and short-duration aircraft campaigns are used to measure all climatologically important properties of the atmosphere. Except for the aircraft campaigns, each deployment operates on a nearly continuous basis to collect high-quality research data sets. Following are the main drivers behind these exercises [131,132]:

1. Determining the accuracy of both solar and infrared radiative transfer calculations for a cloudy atmosphere

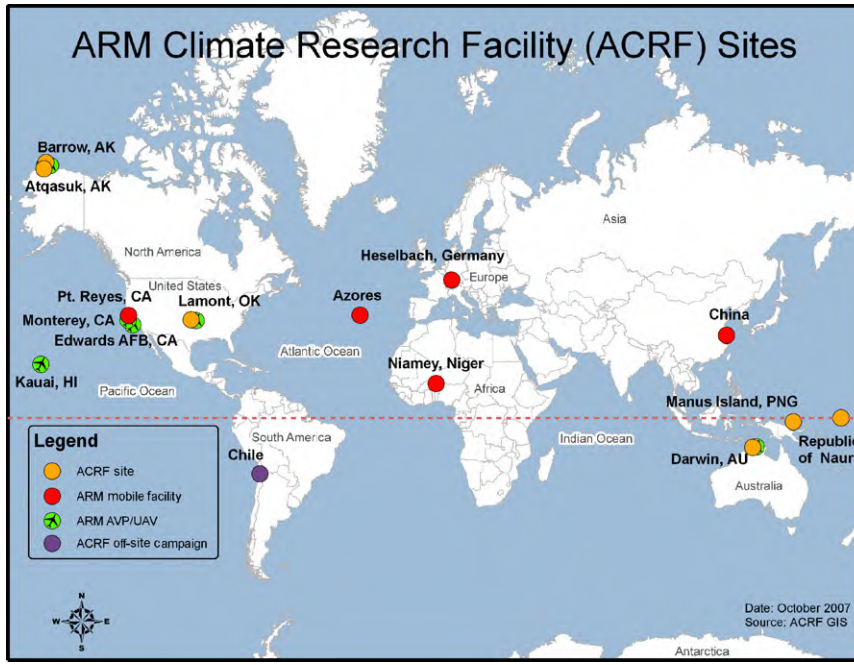

FIG. 16. Location of ARM program sites. The yellow and red markers (M) represent permanent and mobile sites, respectively.

2. Using the knowledge of larger scale atmospheric properties to predict the cloud properties in a column of atmosphere
3. Validating data collected from other sensing experiments as satellite missions

3.3.1 Data-Flow Architecture

As illustrated in Fig. 17, the ACRF data-system architecture is a centrally managed architecture [133]. The ''site data system'' refers to the computing infrastructure located at each site that is responsible for periodically communicating with every local sensor and data-collection system. The heterogeneous data format and communication protocol for different instruments require the data-collection process to be highly resilient and modular. Key challenges in this phase lie in ensuring data integrity and time-synchronization with a universal time reference.

Following collection from sensors, data are routed to the central Data Management Facility located at PNNL. Incoming data for every data source are processed

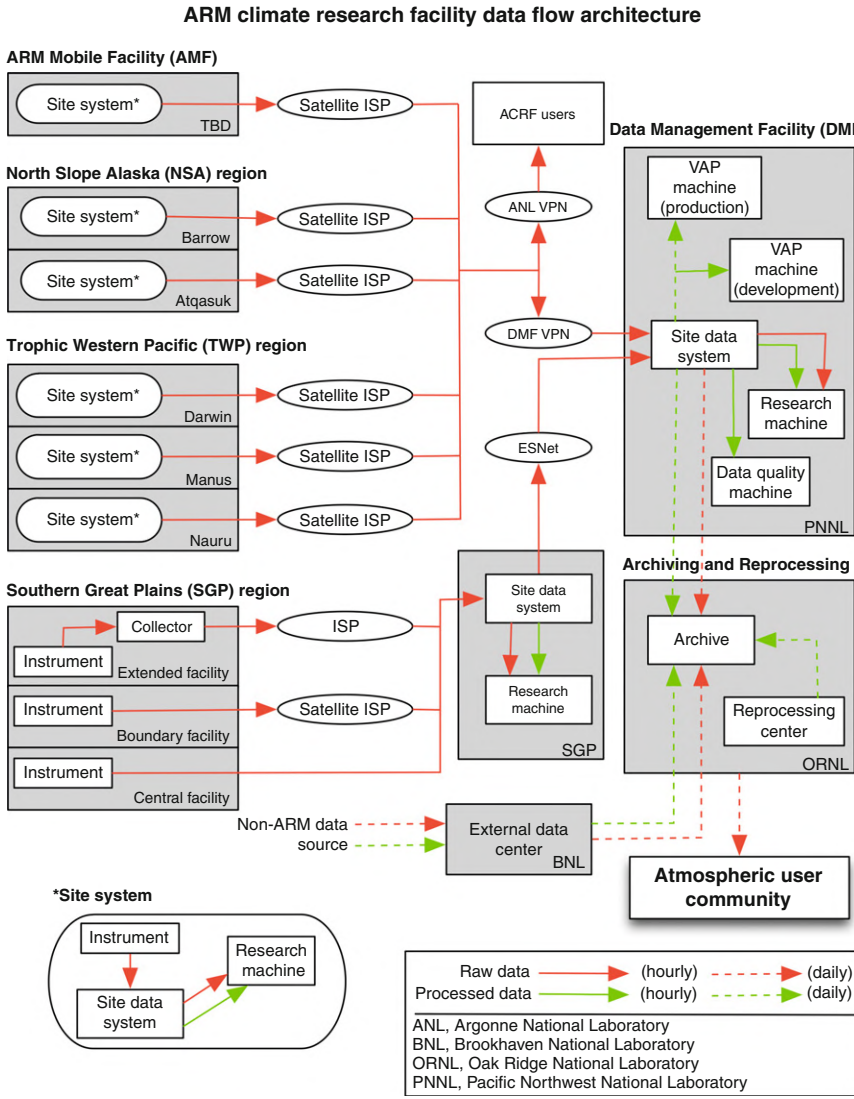

FIG. 17. ARCF data-flow architecture.

hourly by an ingest software suite [133]. The ingest suite is a modular framework that converts native format data files to the ARM standard NetCDF format [134] and performs validations as min–max delta checks and ascending order of timestamps in

data records. Once processed through the ingest suite, the data sets for each data stream pass through a series of downstream processes prior to being transferred to the Data Archive at Oak Ridge National Laboratory. The specific set of downstream processes operating on a data stream depends on the characteristics of the specific data product. Such downstream processes include software for:

- Verifying data integrity
- Archival of files and associated metadata management
- Analysis and visualization for data quality assessment and reporting
- Applying complex algorithms to produce data products with richer scientific content, Value-Added Products (VAPs)

3.3.2 Computing Challenges

As mentioned earlier, the primary focus of the ACRF architecture is to collect high-quality research data sets. Over the years, dramatic reductions in storage costs coupled with improved connectivity with even the remote sites have motivated ARM scientists to maximize the use of sensing capabilities.

However, as the increased observational database improves the representation of clouds and related processes in the scientific world, it also helps identifying the new areas to explore. For example, once the theory for studying atmospheric characteristics at the particular wavelength matures, a next step is to develop a model for the entire frequency spectrum, which subsequently results in increased temporal and frequency domain sampling of the observed phenomena. This advance naturally translates into fielding new research instruments with unique characteristics. Insights gained following the deployment and initial data-collection period lead to analyzing instrument performance, improving measurement accuracy, and developing better retrieval techniques. In this example of data-driven science, data and science enrich each other in a cumulative fashion.

Figure 18 shows the projected storage and data transfer requirements at ACRF through 2015. The rapid growth in data (Fig. 18) presents multifaceted challenges to the ACRF. In the following sections, we present a glimpse of the thrusts at ACRF in four major areas:

1. *Efficient data transfer/storage*. Section 3.3.2.1 describes a data-filtering scheme used for compressing the high-volume spectral data from ARM cloud radars.
2. *Stronger guarantees for data quality*. Section 3.3.2.2 provides the motivation and brief overview of the data quality analysis for one of major data products from ARM.

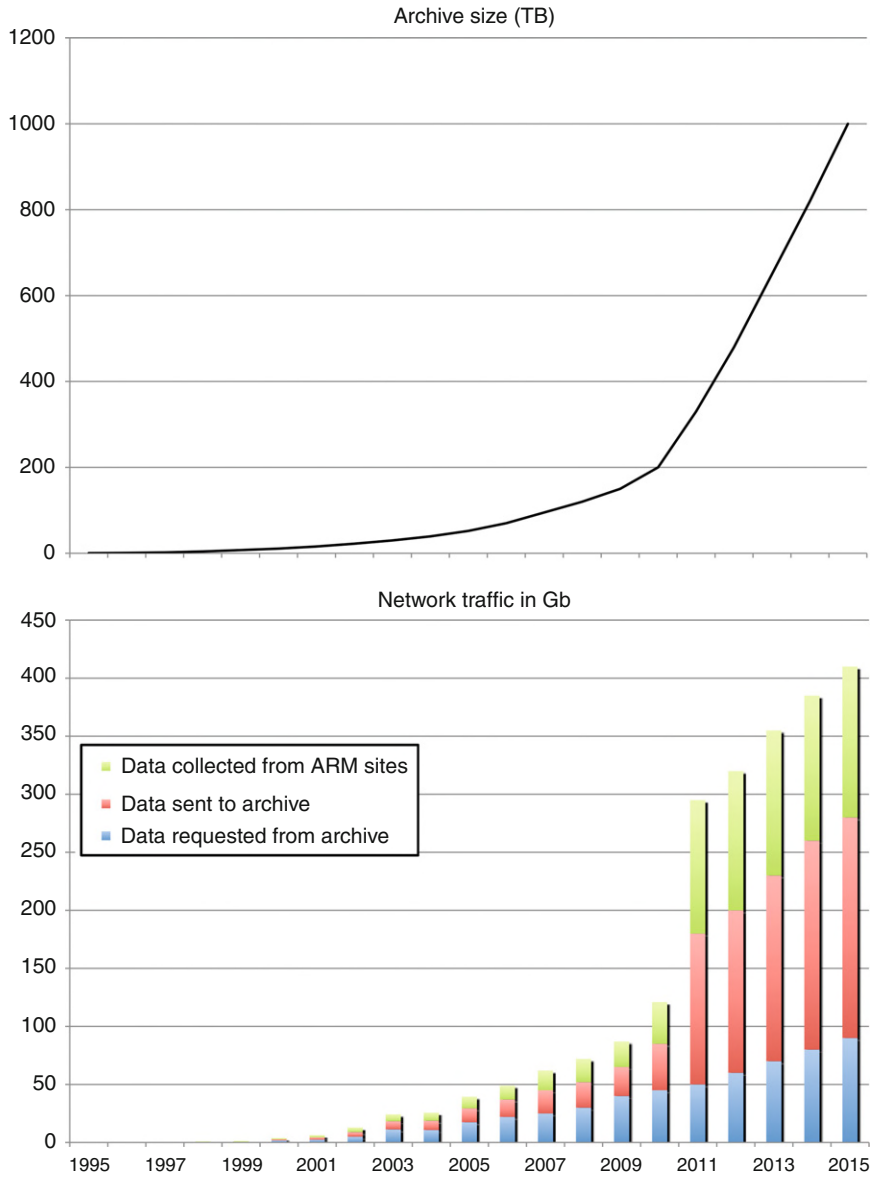

FIG. 18. Projected data transfer and storage requirements for the ACRF.

3. *Reducing time to insight*. Section 3.3.2.3 describes a study focused on the state-of-the-art scientific workflow tools for rapid development of data analysis modules.
4. *Exhaustive and systematic management of metadata*. Section 3.3.2.4 describes the latest efforts in the area of data provenance.

3.3.2.1 Cloud Radar Spectra Processing.

The cloud radars are a key component of the ground-based sensing architecture at ACRF. The latest radar systems continuously record spectral information, providing a very rich data set for deriving physical information at the small scale not previously available in a continuous manner. However, the size of the resulting spectral data files present a significant challenge in transferring and storing the data. The average data volume (include downstream data products) expected from a radar system is around 15 GB/day [135]. The ARM program expects to have 27 operational radar systems over the next 2–3 years that will result in nearly 0.4 TB to be collected daily.

When first challenged with spectral data processing and network-based transfer, investigations into compression techniques for radar spectra were started at ACRF. It is well known that radar data from clear-sky conditions contains low information content.

A cloud detection algorithm developed by Kollias et al. [136] was selected for determining if a spectrum corresponds to clouds or other objects of interest. The algorithm performed the initial masking based on the signal-to-noise ratio and determined an appropriate threshold for a field on a profile-by-profile basis with the Hildebrand and Sekhon [137] technique. A final mask was determined by applying a 3–*x*–3 point majority filter to the first-pass mask (Fig. 19). The spectral algorithm ran onsite and was implemented in interactive data language [138]. The cloud detection algorithm identified data segments with low information content and replaced the samples with a predetermined constant value. The volume of the final data set was observed to be around 5–35% of the original volume depending on the noise content. This technique seemed to work well in practice, although there were some problems. For example, the cloud detection algorithm sometimes missed very ''thin'' clouds. Tuning the algorithm to detect these objects ran the risk of potentially saving spectra that were not meteorologically significant. Currently, this problem is being addressed by capturing additional spectral moments that also serve as statistical summary of the data. An additional engineering challenge involves adapting and implementing this technique for different radar systems that often have very different format and precision characteristics.

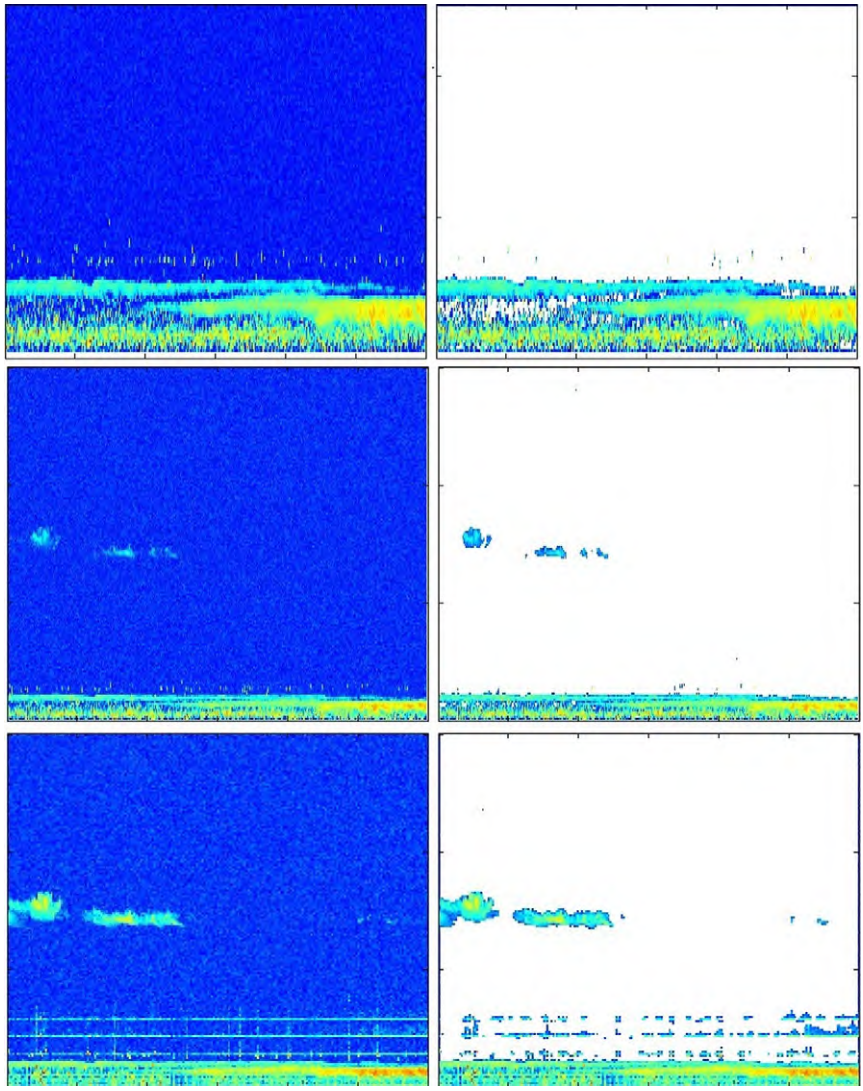

FIG. 19. Spectra profiles before and after filtering.

3.3.2.2 QCRad: A Data Quality Analysis Scheme for Radiation Measurements. Providing detailed characterization of data quality, which is one of the primary responsibilities at ACRF, is achieved by assigning a quality rating to every data record. Lessons learned from advanced statistical experiments, analysis of historical data, and improved visual analytics capabilities contribute toward continuous evolvement of the data-quality-assurance process. The ACRF Data Quality Office at the University of Oklahoma carries out detailed inspection of the daily processed data [139]. For advanced analysis, efforts are undertaken to identify the sources of uncertainty and, for cases in which the data are of sufficient quality, tests are performed to ascertain the presence of biases or precision errors. In this section, we present a brief overview of the QCRad Value-Added Procedure [140] that implements an exhaustive quality control of surface broadband radiometer data. This example demonstrates the complexity of data validation algorithms that are unique for every sensor class.

The QCRad quality-testing methodology involves climatological analyses of surface radiation data to define appropriate limits for trapping unusual data values. In addition to fairly standard min–max limits for the concerned data fields, this methodology makes effective use of knowledge about the cross-correlation between different parameters and derives tighter bounds for performing quality checks. It applies multiple tests to every data value and each progressive test is restrictive or tighter than the previous one.

A separate QC field in the output file accompanies each output data sample. The value of the QC field is set according to the severity of a test failure or infeasibility of a test. A value of zero indicates successful passing of checks, a nonzero value indicates a problem, and higher values suggest an increasing severity of problems with the data. Figure 20A shows a graph of down welling diffuse shortwave irradiance as a function of solar zenith angle. The green, blue, and red lines in the graph refer to maximum limits computed by three techniques. The green line refers to the limits computed from long-term observations of most frequent values for the particular sensor. The blue line defines the limit derived from all historical observations. This accounts for data values that are possible but occur rarely and, thus, might be anomalous data. The red line refers to the globally physically possible limits for surface radiation measurements established by the Baseline Surface Radiation Network. Tests involving the first two levels only annotate the sample with the appropriate code, thus leaving the user with the decision of whether or not to use the data. Any data falling outside the second level of testing is considered an error, in which case the sample value is set to -9999 and the QC field to set appropriately to indicate the test failed. Figure 20B illustrates a case where the limits as indicated by the envelope, for the results of a cross-comparison test are computed. Points shown

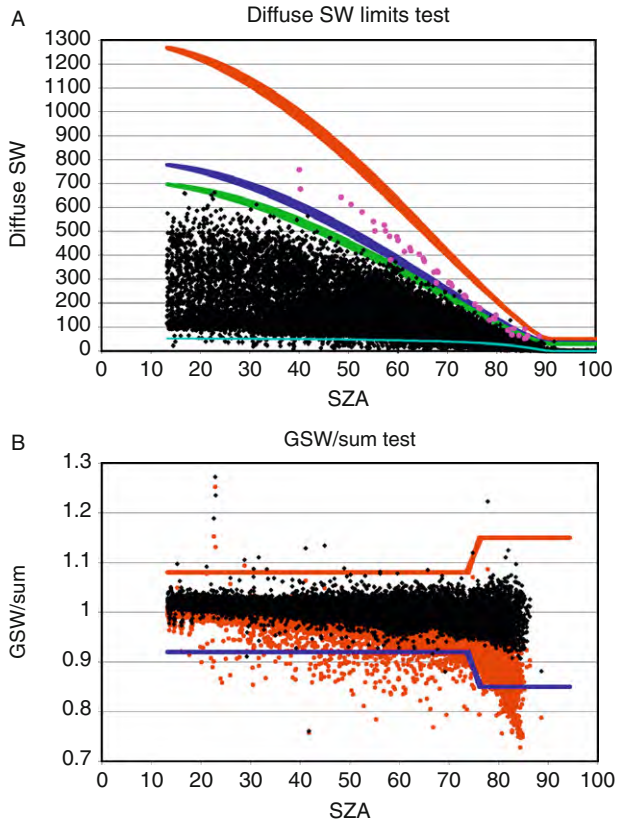

FIG. 20. (A) Example of applying multiple bounds for progressively stricter quality checks. (B) Illustration of an application of data correction algorithm, where black indicates corrected values (images provided by the ARM program).

in red and black represent data values before and after applying a correction algorithm. The effectiveness of the correction process can be clearly seen (Fig. 20B).

The above cases highlight a few problems. To capture the errors in the data, it is important to incorporate the domain knowledge for each and every sensor in the data validation process. Developments of these algorithms or tests often proceed in an iterative and incremental fashion. As the data volume grows, it is important to adapt processing frameworks that can leverage on the processing power of cluster or multicore architectures for quick feedback. Second, it is critically important to

have rapid prototyping tools that abstract away the complexities of DIC and parallel processing and let the user focus more into the scientific pursuits. The next section discusses an investigation in this direction.

3.3.2.3 Rapid Development with Service-Oriented Scientific Workflow Frameworks. In the ACRF, VAPs refer to software suites that perform advanced scientific analysis and generate higher level data sets. Traditionally the VAPs contain serial Fortran or C implementation of various physics algorithms, and have complex input–output dependencies that result in a large monolithic codebase and require manual processing guidance. The complexity of the overall process hinders code reuse and prevents rapid data analysis. For large-scale data processing tasks, the intermediate steps often can be distributed across multiple computing nodes. However, the complexity of partitioning the input data across multiple compute nodes, merging or combining the output results of intermediate steps before proceeding to another computation phase, and dealing with fault tolerance are problems that result in diversion from actual data analysis (Fig. 21). Figure 21A demonstrates the interaction of the MeDICi architecture with the Kepler workflow engine on a VAP. Figure 21B shows the actual workflow of the Broadband Heating Rate Profile (BBHRP) VAP with the embedded parallelism.

To support a more interactive and reusable computing environment, a study was carried out to implement the ARM Program BBHRP VAP [141], in which the Kepler [142,143] and MeDICi technologies [61,144] are used. Kepler is an established Scientific Workflow technology that comes prepacked with a number of components suited for scientific computing. Its ‘‘drag-and-drop’’ design environment supports multiple execution modes and complex control structures that are typical features of VAPs. The MeDICi Integration Framework is a software integration framework that allows a component-based programming model and supports execution on a distributed-processing system. The MeDICi asynchronous messaging-based intercomponent communication framework has been found to be more robust and efficient than the Kepler job-monitoring implementation for a distributed-computing environment; hence, the combination of Kepler and MeDICi were chosen as the underlying development framework. A detailed description of this study was presented by Chase et al. [145]. The following were the main observed benefits of a service-oriented workflow technology for VAP development:

- It allowed splitting the BBHRP application into simple, independent components that can be reused.
- The workflow development environment provides a higher level abstraction of the application.

- Simply changing a configuration can control parallel execution of various phases.
- Hides details of underlying job execution typically performed by complex shell scripts.

The separation of responsibilities, in which a software developer could be tasked with implementing an algorithm (i.e., a component in the workflow) and the domain scientists could design the high-level workflow structure was the most positive outcome of this exercise.

3.3.2.4 Metadata and Data Provenance. Within ARM, metadata include everything from the computing environment, file information (e.g., creation date, last modified date, owner), instrument and algorithm

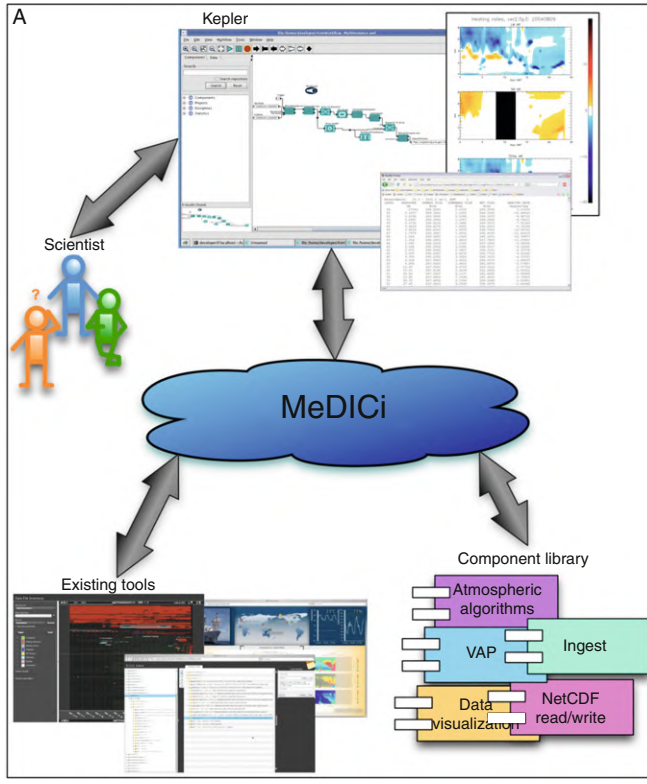

FIG. 21. (Continued)

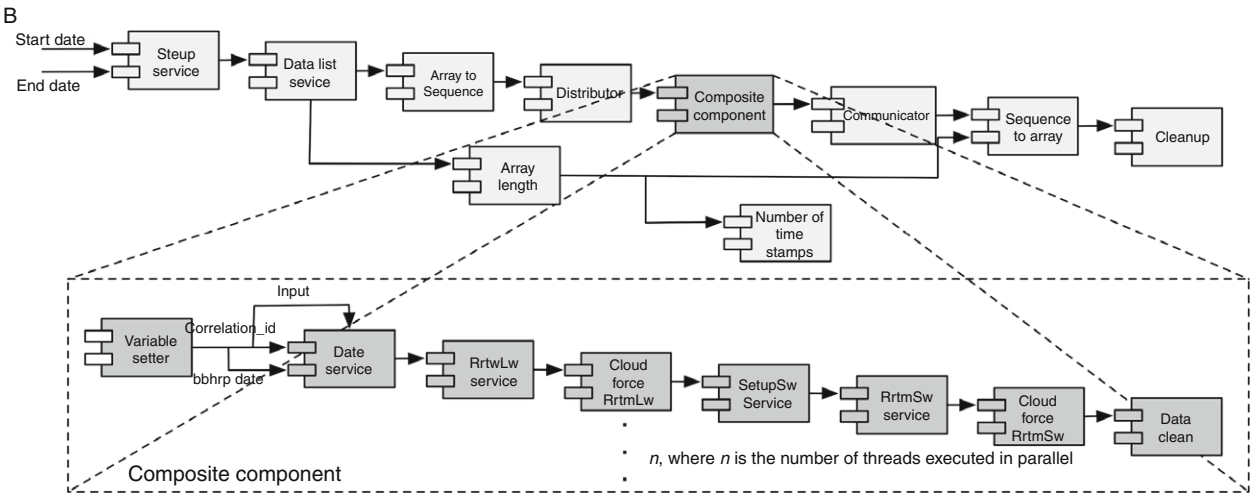

FIG. 21. (A) Interaction between MeDICi architecture and Kepler workflow engine. (B) ACRF VAP workflow.

information, user annotations, and provenance tracking information such as curation history. Provenance is informally defined as ‘‘. . .a causal graph that shows how a result was computed’’ [146]. The positive effect of formal use of provenance is that both producer and consumer communities, motivated by different problems, can make their own assessments about the efficient use of metadata. Without metadata, analysts would be forced to rely on personal knowledge to identify interesting data sets or download climate data in bulk and sift through entire data sets just to find the relevant subset. As the number and size of data sets continue to grow, it becomes challenging to collect useful and informative metadata because the current approaches to gathering the data are invasive and require significant interaction from the scientist or for the developer to have domain knowledge. From an operational standpoint, provenance many times exists partially embedded in the software, reference databases, Web sites, or configuration files and may not be completely available or evident to the scientific community.

3.3.3 ARM Value-Added Products

As stated before, the VAPs provide derived data products through complicated workflows. VAPs use information from sensors, models, algorithms, and other VAPs to derive information of interest that is either impractical or impossible to measure directly or routinely. Areas in which VAPs help fill the unfulfilled measurement needs of ARM include information like cloud microphysics, aerosol properties, atmospheric state, and radiometric properties. The VAPs also are used to improve the quality of existing sensor data, and to help the scientist choose the ‘‘best’’ data when multiple sensors are producing similar data at the same location. As stated before, the file-based standard NetCDF is used in the ARM program for data storage. One reason the NetCDF format was chosen for data storage on the ARM program is the ability NetCDF provides to embed the metadata with the data. This provides users the information they need to analyze the data. A significant user need is to directly disseminate provenance into the VAP output NetCDF file, providing needed information without requiring significant changes to the large body of existing VAP workflows. As the ARM program continues to advance its understanding of atmospheric science, the need for an increasing number and diversity of deployed sensors and VAPs will increase significantly. This increase is expected to affect the amount of data used by the scientists as well as the complexity of VAP interdependency (Fig. 22). Provenance is a significant contributor to the overarching data-driven standard advancing scientific discovery and many of the day-to-day tasks relating to data processing and reprocessing, error detection, and data quality.

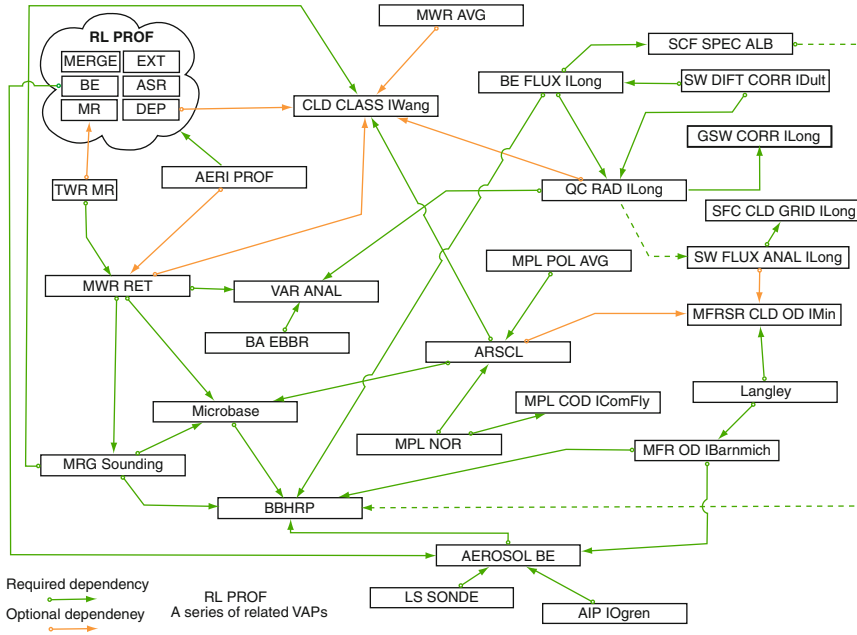

FIG. 22. ACRF VAP dependency graph.

4. Conclusions

This chapter has outlined some of the efforts being carried out at PNNL to advance the state of the art in DIC for biology, cyber security, and atmospheric sciences. The parallel developments at PNNL in DIC for these three fields have provided a synergy that has allowed for rapid progress in these application areas. At the core of this progress has been the transparency of accessing DIC resources through the MeDICi framework.

DIC is evolving rapidly, a transformation driven by the demands of science, engineering, and commerce. In all application areas, issues abound, while solutions lag, partly due to the difficulty of defining the full scope of what is encompassed by the diversity of DIC. Core issues of data-intensive architectures and approaches require a concerted effort in order that progress can be made before we collapse under the burden of our data-intensive world.

Acknowledgement

This work was supported by the Pacific Northwest National Laboratory's LDRD-funded *Data-Intensive Computing Initiative*. Pacific Northwest National Laboratory is operated for the Department of Energy by Battelle under contract DE-AC06-76RLO 1830. PNNL-SA-37658.

References

[1] J.F. Gantz, F.D. Reinsel, C. Chute, W. Schlichting, J. Mcarthur, S. Minton, I. Xheneti, A. Toncheva, A. Manfrediz, IDC—The Expanding Digital Universe: A Forecast of Worldwide Information Growth Through 2010, An IDC White Paper—sponsored by EMC.IDC, Framingham, MA, 2007, 24 pp. Accessed January 11, 2010 at http://www.emc.com/collateral/analyst-reports/expanding-digital-idc-white-paper.pdf.

[2] J.F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, A. Toncheva, The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011, An IDC White Paper—sponsored by EMC.IDC, Framingham, MA, 2008. 16 pp. Accessed January 11, 2010, at http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf.

[3] G. Anthes, Chevron: Where Size Is Opportunity, Computerworld Inc., Framingham, MA, 2006, p. 22.

[4] R. Highfield, Large Hadron Collider: Thirteen Ways to Change the World, 2008. (September 16 Issue), http://www.telegraph.co.uk.

[5] S.H. Muggleton, 2020 Computing: exceeding human limits, Nature 440 (2006) 409–410.

[6] W. Strand, Climate model data management—future challenges, Geophys. Res. Abs. 8 (2006) 05249. Accessed June 22, 2009, at http://www.cosis.net/abstracts/EGU06/05249/EGU06-J-05249.pdf?PHPSESSID=2637d92889c27b41f86472d745c8ad36.

[7] S.J. Williamson, D.B. Rusch, S. Yooseph, A.L. Halpern, K.B. Heidelberg, J.I. Glass, C. Andrews-Pfannkoch, D. Fadrosh, C.S. Miller, G. Sutton, M. Frazier, J.C. Venter, The Sorcerer II Global Ocean Sampling Expedition: Metagenomic characterization of viruses within aquatic microbial samples, PLoS ONE 3 (1) (2008) e1456.

[8] A.-L. Barabasi, Z.N. Oltvai, Network biology: understanding the cell's functional organization, Nat. Rev. Genet. 5 (2004) 101–113.

[9] A. Szalay, J. Gray, 2020 Computing: science in an exponential world, Nature 440 (2006) 413–414.

[10] J.J. Thomas, K.A. Cook, Illuminating the Path: The Research and Development Agenda for Visual Analytics, IEEE Computer Society, Los Alamitos, CA, 2005.

[11] R.T. Kouzes, G.A. Anderson, S.T. Elbert, I. Gorton, D. Gracio, The changing paradigm of data intensive computing, Computer 42 (1) (2009) 26–34.

[12] Price Waterhouse Coopers, Global entertainment and media outlook: 2007–2011, Price Waterhouse Coopers Report. 2007.

[13] R.W. Crandall, J.G. Sidak, Video games: serious business for America's economy, Entertainment Software Association Report. 2006. Available at http://ssrn.com/abstract=969728.

[14] L. Flath, M. Kartz, R. Frank, Utilizing commercial graphics processors in the real-time geo-registration of streaming high-resolution imagery, in: GP^2 Workshop, Chapel Hill, NC, August 7–8, 2004.

[15] J. Johnson, R. Frank, S. Vaidya, The evaluation of GPU-based programming environments for knowledge discovery, in: HPEC, 2004.

[16] D. Luebke, M. Harris, J. Kruger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, A. Lefohn, GPGPU: general purpose computation on graphics hardware, in: SIGGRAPH'04: ACM

SIGGRAPH 2004 Course Notes, ACM Press, New York, NY, 2004. http://dx.doi.org/10.1145/1103900.1103933.
[17] R. Fernando (Ed.), GPU Gems, Addison-Wesley Professional, New York, NY, 2004.
[18] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, T. Purcell, A survey of general-purpose computation on graphics hardware, Comput. Graph. Forum 26 (1) (2007) 80–113.
[19] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with CUDA, ACM Queue 6 (2) (2008) 40–53.
[20] J. Chong, Y. Yi, A. Faria, N. Satish, K. Keutzer, Data-parallel large vocabulary continuous speech recognition on graphics processors, in: Proceedings of the 1st Annual Workshop on Emerging Applications and Many Core Architecture (EAMA), 2008.
[21] D. Steinkraus, I. Buck, P. Simard, Using GPUs for machine learning algorithms, in: Proceedings of the 5th International Conference on Document Analysis and Recognition (ICDAR), 2005.
[22] N.K. Govindaraju, M. Lin, D. Manocha, GPGP: general purpose computations using graphics processors, in: Proceedings of the 9th Annual Workshop on High Performance Embedded Computing, 2005.
[23] Y. Liu, W. Huang, J. Johnson, S. Vaidya, GPU accelerated Smith-Waterman, in: International Conference on Computational Science, 2006, 4 vols, pp. 188–195.
[24] B. Flachs, S. Asano, S.H. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, et al., A streaming processing unit for a CELL processor, in: IEEE International Solid-State Circuits Conference, 2005.
[25] H.P. Hofstee, Power efficient processor architecture and the cell processor, in: HPCA, 2005.
[26] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, K. Yelick, The potential of the cell processor for scientific computing, in: Conference on Computing Frontiers, Ischia, Italy, 2005.
[27] F. Blagojevic, D.S. Nikolopoulos, A. Stamatakis, C.D. Antonopoulous, Dynamic multigrain parallelization on the cell broadband engine, in: Principles and Practice of Parallel Programming, San Jose, CA, 2007.
[28] Y. Liu, H. Jones, S. Vaidya, M. Perrone, B. Tydlitát, A.K. Nanda, Speech recognition systems on the Cell Broadband Engine processor, IBM J. Res. Dev. 51 (5) (2007) 583–592.
[29] D.P. Scarpazza, O. Villa, F. Petrini, Peak-performance DFA-based string matching on the cell processor, in: IEEE International Parallel and Distributed Processing Symposium, 2007 (IPDPS 2007), (2007), pp. 1–8. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4228362.
[30] O. Villa, D.P. Scarpazza, F. Petrini, J.F. Peinador, Challenges in mapping graph exploration algorithms on advanced multi-core processors, in: Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS), Long Beach, CA, March 26–30, 2007.
[31] L.-K. Liu, Q. Liu, A. Natsev, K.A. Ross, J.R. Smith, A.L. Varbanescu, Digital media indexing on the cell processor, in: Proceedings of the 2007 IEEE International Conference on Multimedia and Expo, (2007), pp. 1866–1869. http://dx.doi.org/10.1109/ICME.2007.4285038.
[32] C.M. Kastner, G.A. Covington, A.A. Levine, J.W. Lockwood, HAIL: a hardware-accelerated algorithm for language identification, in: International Conference on Field Programmable Logic and Applications, August 24–26, 2005, pp. 499–504. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1515771&isnumber=32467.
[33] K. Siozios, D. Soudris, A novel methodology for exploring interconnection architectures targeting 3-D FPGAs, in: 3rd HiPEAC Workshop on Reconfigurable Computing (WRC), Paphos, Cyprus, 2009.
[34] J.D. Buhler, J.M. Lancaster, A.C. Jacob, D. Roger, Mercury BLASTN: faster DNA sequence comparison using a streaming hardware architecture, in: Proceedings of the 3rd Annual Reconfigurable Systems Summer Institute, Urbana-Champaign, IL, 2007.

[35] I. Sourdis, D. Pnevmatikatos, Fast, large-scale string match for a 10Gbps FPGA-based network intrusion detection system, in: Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL 2003), Lisbon, Portugal, September 1–3, 2003.
[36] S. Parsons, D.E. Taylor, High Speed Processing of Financial Information Using FPGA Devices, 2008. US Patent Application US 2007/0243675 A1.
[37] W. Anderson, P. Briggs, C.S. Hellberg, D.W. Hess, A. Khokhlov, M. Lanzagorta, R. Rosenberg, Naval Research Laboratory, Early experience with scientific programs on the Cray MTA-2., in: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, 2003, p. 46.
[38] J. Feo, D. Harper, S. Kahan, P. Konecny, ELDORADO, in: N. Bagherzadeh, M. Valero, A. Ramirez (Eds.), Proceedings of the 2nd Conference on Computing Frontiers, ACM Press, Ischia, Italy, 2005, pp. 28–34.
[39] P. Kongetira, K. Aingaran, K. Olukotun, Niagara: a 32-way multithreaded SPARC processor, IEEE Micro 25 (2) (2005) 21–29.
[40] U.G. Nawathe, M. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar, H. Park, An 8-core, 64-thread, 64-bit power efficient SPARC SOC (Niagara2), in: Proceedings of the 2007 International Symposium on Physical Design, Austin, TX, ACM Press, New York, NY, 2007, pp. 2–2.
[41] D. Chavarria-Miranda, A. Marquez, J. Nieplocha, K. Maschhoff, C. Scherrer, Early experience with out-of-core applications on the Cray XMT, in: Proceeding of MTAAP'08: Workshop on Multithreaded Architectures and Applications, IEEE, Miami, FL, 2008.
[42] C. Scherrer, N. Beagley, J. Nieplocha, A. Marquez, J. Feo, D. Chavarria-Miranda, Probability convergence in a multithreaded counting application, in: IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 1–5.
[43] Cray, Cray XD-1, 2009. Accessed January 1, 2010, at http://www.cray.com/products/Legacy.aspx.
[44] E. Riedel, G. Gibson, C. Faloutsos, Active storage for large-scale data mining and multimedia, in: Proceedings of the 24th International Conference on Very Large Databases (VLDB'98), New York, NY, August 24–27, 1998.
[45] A. Yoo, S. Kohn, T. Brugger, I. Kaplan, S. Pingry, Searching a massive semantic graph on Netezza performance server, in: Conference on Supercomputing, Tampa, FL, 2006.
[46] G.S. Davidson, K.W. Boyack, R.A. Zacharski, S.C. Helmreich, J.R. Cowie, Data-Centric Computing with the Netezza Architecture, Sandia Technical Report. 2006.
[47] M. Gokhale, J. Cohen, W.A. Yoo, M. Miller, A. Jacob, C. Ulmer, R. Pearce, Hardware technologies for high-performance data-intensive computing, Computer 41 (4) (2008) 60–68.
[48] C. Ulmer, M. Gokhale, Threading opportunities in high-performance flash-memory storage, in: High Performance Embedded Computing (HPEC) Workshop, 2008.
[49] D. Ajwani, I. Malinger, U. Meyer, S. Toledo, Characterizing the performance of flash memory storage devices and its impact on algorithm design, in: Proceedings of the 7th International Workshop on Experimental Algorithms (WEA), Provincetown, MA, 2008, pp. 208–219.
[50] M.W. Berry, S.T. Dumais, G.W. O'Brien, Using linear algebra for intelligent information retrieval, SIAM Rev. 37 (4) (1995) 573–595.
[51] M. Damashek, Gauging similarity with n-grams; language-independent categorization of text, Science 267 (1995) 843–848.
[52] G. Salton, C. Yang, A. Wong, A vector space model for automatic indexing, Commun. ACM 18 (11) (1975) 613–620.
[53] G. Schmidt, T. Ströhlein, Relations and Graphs, Springer-Verlag, Berlin, 1993.
[54] P.D. Whitney, D. Cox, D.S. Daly, H.P. Foote, D.L. McQuerry, J.M. Sloughter, Toward the routine analysis of diverse data types, J. Comput. Graph. Stat. 12 (4) (2003) 915–926.

[55] E. Wegman, Huge data sets and the frontiers of computational feasibility, J. Comput. Graph. Stat. 4 (4) (1995) 281–295.
[56] I. Gorton, J. Chase, A.S. Wynne, J. Almquist, A. Chappell, Services + components = data intensive scientific workflow applications with MeDICi, in: Proceedings of the International Symposium on Component-Based Software Engineering, Springer-Verlag, Berlin, 2009.
[57] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, Concurr. Comput.: Pract. Exp. 18 (10) (2006) 1039–1065.
[58] Information Week, August 6, 2007.
[59] A. Szalay, J. Gray, J. Vandenberg, Petabyte scale data mining: dream or reality? in: SPIE Astronomy Telescopes and Instruments, Waikoloa, HI, 2003.
[60] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (January 2008) 107–113.
[61] I. Gorton, A.S. Wynne, J.P. Almquist, J. Chatterton, The MeDICi integration framework: a platform for high performance data streaming applications, in: WICSA 2008, 7th IEEE/IFIP Working Conference on Software Architecture, 2008, pp. 95–104.
[62] BPEL, 2009. Accessed at http://www.softcare.com/whitepapers/wp_whatis_bpel.php.
[63] A.R. Shah, M. Singhal, T.D. Gibson, C. Sivaramakrishnan, K.M. Waters, I. Gorton, An extensible, scalable architecture for managing bioinformatics data and analyses, in: Proceedings of the 4th IEEE International Conference on eScience'08, Indianapolis, IN, 2008, pp. 190–197.
[64] R.H. Waterston, E.S. Lander, J.E. Sulston, On the sequencing of the human genome, Proc. Natl. Acad. Sci. 99 (6) (2002) 3712–3716.
[65] S. Yooseph, G. Sutton, D.B. Rusch, A.L. Halpern, S.J. Williamson, K. Remington, J.A. Eisen, K.B. Heidelberg, G. Manning, W. Li, L. Jaroszewski, P. Cieplak, et al., The Sorcerer II Global Ocean Sampling Expedition: expanding the universe of protein families, PLoS Biol. 5 (3) (2007) e16.
[66] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, Basic local alignment search tool, J. Mol. Biol. 215 (3) (1990) 403–410.
[67] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nucleic Acids Res. 25 (17) (1997) 3389–3402.
[68] C.S. Oehmen, J. Nieplocha, ScalaBLAST: a scalable implementation of BLAST for high performance data-intensive bioinformatics analysis, IEEE Trans. Parallel Distrib. Syst. 17 (8) (2006) 740–749.
[69] V.M. Markowitz, F. Korzeniewski, K. Palaniappan, E. Szeto, G. Werner, A. Padki, X. Zhao, I. Dubchak, P. Hugenholtz, I. Anderson, A. Lykidis, K. Mavrommatis, et al., The integrated microbial genomes (IMG) system, Nucleic Acids Res. 34 (Database-Issue) (2006) 344–348.
[70] V.M. Markowitz, N. Ivanova, E. Szeto, K. Palaniappan, K. Chu, D. Dalevi, I.-M.A. Chen, Y. Grechkin, I. Dubchak, I. Anderson, A. Lykidis, K. Mavrommatis, et al., IMG/M: a data management and analysis system for metagenomes, Nucleic Acids Res. 36 (Database-Issue) (2008) 534–538.
[71] A.R. Shah, V.M. Markowitz, C.S. Oehmen, High-throughput computation of pairwise sequence similarities for multiple genome comparisons using ScalaBLAST, in: IEEE/NIH Life Science Systems and Applications Workshop (LISA 2007), Bethesda, MD, November 8–9, 2007, pp. 89–91.
[72] C.S. Oehmen, W.R. Cannon, Bringing high performance computing to the biologist's workbench: approaches, applications and challenges, SciDAC 2008: J. Phys.: Conf. Ser. 125 (2008) 012052.
[73] A.R. Shah, C.S. Oehmen, B.-J. Webb-Robertson, SVM-Hustle—an iterative semi-supervised machine learning approach for pairwise protein remote homology detection, Bioinformatics 24 (6) (2008) 783–790.

[74] B.J.M. Webb-Robertson, C.S. Oehmen, A.R. Shah, A feature vector integration approach for a generalized support vector machine pairwise homology algorithm, Comput. Biol. Chem. 32 (6) (2008) 458–461.
[75] R. Aebersold, M. Mann, Mass spectrometry-based proteomics, Nature 422 (6928) (2003) 198–207.
[76] B. Bogdanov, R.D. Smith, Proteomics by FTICR mass spectrometry: top down and bottom up, Mass Spectrom. Rev. 24 (2) (2005) 168–200.
[77] W.H. McDonald, J.R. Yates III, Shotgun proteomics: integrating technologies to answer biological questions, Curr. Opin. Mol. Ther. 5 (3) (2003) 302–309.
[78] S. Sechi, Y. Oda, Quantitative proteomics using mass spectrometry, Curr. Opin. Chem. Biol. 7 (1) (2003) 70–77.
[79] D.A. Wolters, M.P. Washburn, J.R. Yates III, An automated multidimensional protein identification technology for shotgun proteomics, Anal. Chem. 73 (23) (2001) 5683–5690.
[80] C.C. Wu, M.J. MacCoss, K.E. Howell, J.R. Yates III, A method for the comprehensive proteomic analysis of membrane proteins, Nat. Biotechnol. 21 (5) (2003) 532–538.
[81] S. Tanner, H. Shu, A. Frank, L.C. Wang, E. Zandi, M. Mumby, P.A. Pevzner, V. Bafna, InsPecT: identification of post-translationally modified peptides from tandem mass spectra, Anal. Chem. 77 (14) (2005) 4626–4639.
[82] Y. Chen, S. Yu, M. Leng, Parallel sequence alignment algorithm for clustering system, in: Knowledge Enterprise: Intelligent Strategies in Product Design, Manufacturing and Management, vol. 207, Springer-Verlag, Boston, MA, 2006, pp. 311–321.
[83] L. Pasa-Tolic, C. Masselon, R.C. Barry, Y. Shen, R.D. Smith, Proteomic analyses using an accurate mass and time tag strategy, BioTechniques 37 (4) (2004) 621–624, 626–636.
[84] J.S. Zimmer, M.E. Monroe, W.J. Qian, R.D. Smith, Advances in proteomics data analysis and display using an accurate mass and time tag approach, Mass Spectrom. Rev. 25 (2006) 450–482.
[85] K. Petritis, L.J. Kangas, B. Yan, E.F. Strittmatter, M. Monroe, W. Qian, J.N. Adkins, R.J. Moore, Y. Xu, M.S. Lipton, D.G. Camp II, R.D. Smith, Improved peptide elution time prediction for reversed-phase liquid chromatography-MS by incorporating peptide sequence information, Anal. Chem. 78 (14) (2006) 5026–5039.
[86] N. Jaitly, M.E. Monroe, V.A. Petyuk, T.R. Clauss, J.N. Adkins, R.D. Smith, Robust algorithm for alignment of liquid chromatography–mass spectrometry analyses in an accurate mass and time tag data analysis pipeline, Anal. Chem. 78 (21) (2006) 7397–7409.
[87] R. Craig, R.C. Beavis, A method for reducing the time required to match protein sequences with tandem mass spectra, Rapid Commun. Mass Spectrom. 17 (2) (2003) 2310–2316.
[88] J.K. Eng, A.L. McCormack, J.R. Yates III, An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database, J. Am. Soc. Mass Spectrom. 5 (11) (1994) 976–989.
[89] I. Beer, E. Barnea, T. Ziv, A. Admon, Improving large-scale proteomics by clustering of mass spectrometry data, Proteomics 4 (4) (2004) 950–960.
[90] R. Craig, J.C. Cortens, D. Fenyo, R.C. Beavis, Using annotated peptide mass spectrum libraries for protein identification, J. Proteome Res. 5 (8) (2006) 1843–1849.
[91] J.R. Yates III, S.F. Morgan, C.L. Gatlin, P.R. Griffin, J.K. Eng, Method to compare collision-induced dissociation spectra of peptides: potential for library searching and subtractive analysis, Anal. Chem. 70 (17) (1998) 3557–3565.
[92] D.M. Horn, R.A. Zubarev, F.W. Lafferty, Automated reduction and interpretation of high resolution electrospray mass spectra of large molecules, J. Am. Soc. Mass Spectrom. 11 (4) (2000) 320–332.
[93] A.L. Rockwood, S.L. Van Orden, R.D. Smith, Rapid calculation of isotope distributions, Anal. Chem. 67 (15) (1995) 2699–2704.

[94] M.W. Senko, S.C. Beu, F.W. McLafferty, Automated assignment of charge states from resolved isotopic peaks for multiply charged ions, J. Am. Soc. Mass Spectrom. 6 (4) (1995) 52–56.
[95] M.W. Senko, S.C. Beu, F.W. McLafferty, Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions, J. Am. Soc. Mass Spectrom. 6 (1995) 229–233.
[96] M.E. Monroe, N. Tolic, N. Jaitly, J.L. Shaw, J.N. Adkins, R.D. Smith, VIPER: an advanced software package to support high-throughput LC-MS peptide identification, Bioinformatics 23 (15) (2007) 2021–2023.
[97] I. Bogdan, D. Coca, J. Rivers, R.J. Beynon, Hardware acceleration of processing of mass spectrometric data for proteomics, Bioinformatics 23 (6) (2007) 724–731.
[98] W.R. Cannon, K.H. Jarman, B.-J. Webb-Robertson, D.J. Baxter, C.S. Oehmen, K.D. Jarman, A. Heredia-Langner, K.J. Auberry, G.A. Anderson, Comparison of probability and likelihood models for peptide identification from tandem mass spectrometry data, J. Proteome Res. 4 (5) (2005) 1687–1698.
[99] R. Craig, R.C. Beavis, TANDEM: matching proteins with tandem mass spectra, Bioinformatics 20 (9) (2004) 1466–1467.
[100] R. Aebersold, L. Anderson, R. Caprioli, B. Druker, L. Hartwell, R. Smith, Perspective: a program to improve protein biomarker discovery for cancer, J. Proteome Res. 4 (2005) 1104–1109.
[101] A. Alaiya, M. Al-Mohanna, S. Linder, Clinical cancer proteomics: promises and pitfalls, J. Proteome Res. 4 (2005) 1213–1222.
[102] K.S. Anderson, J. LaBaer, The sentinel within: exploiting the immune system for cancer biomarkers, J. Proteome Res. 4 (2005) 1123–1133.
[103] M.A. Gillette, D.R. Mani, S.A. Carr, Place of pattern in proteomic biomarker discovery, J. Proteome Res. 4 (4) (2005) 1143–1154.
[104] J.M. Jacobs, J.N. Adkins, W.-J. Qian, T. Liu, Y. Shen, D.G. Camp II, R.D. Smith, Utilizing human blood plasma for proteomic biomarker discovery, J. Proteome Res. 4 (4) (2005) 1073–1085.
[105] S. Srivastava, R.-G. Srivastava, Proteomics in the forefront of cancer biomarker discovery, J. Proteome Res. 4 (4) (2005) 1098–1103.
[106] S. Srivastava, M. Verma, R. Gopal-Srivastava, Proteomic maps of the cancer-associated infectious agents, J. Proteome Res. 4 (4) (2005) 1171–1180.
[107] J. Villanueva, J. Philip, C.A. Chaparro, Y. Li, R. Toledo-Crow, Correcting common errors in identifying cancer-specific serum peptide signatures, J. Proteome Res. 4 (2005) 1060–1072.
[108] F. Vitzthum, F. Behrens, S.L. Anderson, J.H. Shaw, Proteomics: from basic research to diagnostic application. A review of requirements & needs, J. Proteome Res. 4 (2005) 1086–1097.
[109] E.F. Petricoin, A.M. Ardekani, B.A. Hitt, P.J. Levine, V.A. Fusaro, S.M. Steinberg, G.B. Mills, C. Simone, D.A. Fishman, E.C. Kohn, L.A. Liotta, Use of proteomic patterns in serum to identify ovarian cancer, Lancet 359 (9306) (2002) 572–577.
[110] E.P. Diamandis, Proteomic patterns in serum and identification of ovarian cancer—reply, Lancet 360 (9327) (2002) 170–171.
[111] M. Elwood, Proteomic patterns in serum and identification of ovarian cancer—reply, Lancet 360 (9327) (2002) 170.
[112] B. Rockhill, Proteomic patterns in serum and identification of ovarian cancer—reply, Lancet 360 (9327) (2002) 169–171.
[113] B.H. Clowers, W.F. Siems, H.H. Hill, S.M. Massick, Hadamard transform ion mobility spectrometry, Anal. Chem. 78 (1) (2006) 44–51.
[114] J.R. Kimmel, Continuous, Multiplexed Time-of-Flight Mass Spectrometry of Electrosprayed Ions, 2004. Ph.D. Thesis, Stanford University, pp. 18–46.

[115] Symantec, M. Fossi, E. Johnson, T. Mach (Eds.), Symantec Global Internet Security Threat Report, vol. XIV, Symantec, Cupertino, CA, 2009.
[116] J. Bigham, D. Gamez, L. Ning, Safeguarding SCADA systems with anomaly detection, in: Proceedings of the 2nd International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS 2003), Lecture Notes in Computer Science, 2776, 2003, pp. 171–182.
[117] L. Feng, X.H. Guan, S. Guo, Y. Gao, P. Liu, Predicting the intrusion intentions by observing system call sequences, Comput. Secur. 23 (3) (2004) 241–252.
[118] S. Noel, M. Jacobs, P. Kalapa, S. Jajodia, Multiple coordinated views for network attack graphs, in: IEEE Workshop on Visualization for Computer Security 2005 (VizSEC 05), Minneapolis, MN, 2005.
[119] P. Saraiya, C. North, K. Duca, Visualizing biological pathways: requirements analysis, systems evaluation and research agenda, Inform. Visual. 4 (3) (2005) 191–205.
[120] H. Zhuge, X. Shi, Fighting epidemics in the information and knowledge age, Computer 36 (10) (2003) 114–116.
[121] S. Krasser, G. Conti, J. Grizzard, J. Gribschaw, H. Owen, Real-time and forensic network data analysis using animated and coordinated visualization, in: IEEE Workshop of Information Assurance and Security, West Point, NY, 2005.
[122] L. Paulson, Researchers develop network-security visualization tools, Computer 37 (4) (2004) 17–18.
[123] G. Conti, K. Abdullah, J. Grizzard, J. Stasko, J.A. Copeland, M. Ahamad, H.L. Owen, C. Lee, Countering security information overload through alert and packet visualization, IEEE Comput. Graph. Appl. 26 (2) (2006) 60–70.
[124] K. Kafadar, E.J. Wegman, Visualizing ''typical'' and ''exotic'' Internet traffic data, Comput. Stat. Data Anal. 50 (12) (2006) 3721–3743.
[125] E.J. Wegman, D. Marchette, On some techniques for streaming data: a case study of Internet packet headers, J. Comput. Graph. Stat. 12 (4) (2003) 893–914.
[126] A. D'Amico, K. Whitley, The real work of computer network defense analysts, in: VizSEC 2007: Proceedings of the Workshop on Visualization for Computer Security, Springer-Verlag, Sacramento, CA, 2008, pp. 19–37.
[127] W. Pike, C. Scherrer, S. Zabriskie, Putting security in context: visual correlation of network activity with real-world information, in: VizSEC 2007: Proceedings of the Workshop on Visualization for Computer Security, Springer-Verlag, Sacramento, CA, 2008, pp. 203–220.
[128] D. Chavarria-Miranda, A. Marquez, J. Nieplocha, K. Maschhoff, C. Scherrer, Early experience with out-of-core applications on the Cray XMT, in: IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), April 14–18, 2008.
[129] A. Moore, M.S. Lee, Cached sufficient statistics for efficient machine learning with large datasets, J. Artif. Intell. Res. 8 (1998) 67–91.
[130] Atmospheric Radiation Measurement Climate Research Facility (ACRF), ARM Annual Report. 2008. Technical report available from U.S. Department of Energy as DOE/SC-ARM-0706.
[131] T.P. Ackerman, The Role of Global Observations for Climate and Other Applications, DOE/SC-ARM/TR-067Pacific Northwest National Laboratory, Richland, WA, 2005.
[132] Atmospheric Radiation Measurement Program (ARM), Report on the ARM Climate Research Facility Expansion Workshop, DOE/SC-ARM-0707, Pacific Northwest National Laboratory for the U.S. Department of Energy, Richland, WA, 2007.
[133] M.C. Macduff, R.C. Eagan, ACRF Data Collection and Processing Infrastructure, DOE/SC-ARM/TR-046. Pacific Northwest National Laboratory, Richland, WA, 2004.

[134] NetCDF, 2009. Accessed at http://www.unidata.ucar.edu/software/netcdf/.
[135] K. Widener, K. Johnson, ARM cloud radars—a year in review and a look to the future, in: Proceedings of 17th ARM Science Team Meeting, Monterey, CA, 2007.
[136] P. Kollias, M.A. Miller, E.P. Luke, K.L. Johnson, E.E. Clothiaux, K.P. Moran, K.B. Widener, B.A. Albrecht, The atmospheric radiation measurement program cloud profiling radars: second-generation sampling strategies, processing, and cloud data products, J. Atmos. Ocean. Technol. 24 (7) (2007) 1199–1214.
[137] P.H. Hildebrand, R.S. Sekhon, Objective determination of the noise level in Doppler spectra, J. Appl. Meteor. 13 (7) (1974) 808–811.
[138] IDL, Interactive Data Language. 2009. Accessed June 29, 2009, at http://www.ittvis.com.
[139] R.A. Peppler, K.E. Kehoe, K.L. Sonntag, C. Bahrmann, S. Richardson, S. Christensen, R. McCord, K. Doty, R. Wagener, R. Eagan, J. Liljegren, B. Orr, et al., Quality Assurance of ARM Program Climate Research Facility Data, DOE/SC-ARM/TR-082, Pacific Northwest National Laboratory, Richland, WA, 2008.
[140] C.N. Long, Y. Shi, An automated quality assessment and control algorithm for surface radiation measurements, Open Atmos. Sci. J. (TOASJ) 2 (2008) 23–37.
[141] Value-Added Product (VAP), 2009. Accessed January 1, 2010, at http://www.arm.gov/data/vaps.
[142] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, S. Mock, Kepler: an extensible system for design and execution of scientific workflows, in: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04), Santorini Island, Greece, June 21–23, 2004.
[143] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, S. Mock, The Kepler Project, 2008. Accessed June 23, 2009, at https://kepler-project.org/.
[144] MeDICi: Middleware for Data-Intensive Computing, 2009. Accessed June 29, 2009, at http://medici.pnl.gov/.
[145] C. Jared, G. Ian, et al., Kepler + MeDICi Service-Oriented Scientific Workflow Applications, Proceedings of the 2009 Congress on Services - I, IEEE Computer Society, 2009.
[146] J. Cheney, Principles of Provenance, University of Edinburgh, Edinburgh, UK, 2009. Accessed June 23, 2009, at http://www.beliefproject.org/events/brainstormings/brainstorming1/ppt/james-cheney-principles-of-provenance.pdf.

Pitfalls and Issues of Manycore Programming

AMI MAROWKA

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Abstract

The transition from sequential computing to parallel computing represents the next turning point in the way software engineers design and write software. The addition of more processing cores has emerged as the primary way to boost the computing power of microprocessors, but first the research community has to overcome certain hardware and software challenges along the way to on-chip scalable systems.

The primary consequence is that applications will increasingly need to be parallelized to fully exploit processor throughput gains that are now becoming available. However, parallel code is more difficult to write than that of serial code. Writing applications in a way that permits different parts of a computing task be divided up and executed simultaneously across multiple cores is not new. Efforts today focus on translating the knowledge of building off-chip supercomputing based on multiprocessors to on-chip hypercomputing based on multicore processors. While the software may present the biggest challenge, there are also hardware changes that need to be made to overcome issues such as scalability and portability.

The aim of this chapter is to explain the primary difficulties and issues of manycore programming. Firstly, the unsolved problem of the parallel computation model and its implications for issues such as portability and performance prediction is discussed. Secondly, the obstacles incurred by parallel hardware architectures on software development are covered. Thirdly, the main traps and pitfalls of multicore programming are addressed. Finally, the human factor in the success of the parallel revolution is presented.

ADVANCES IN COMPUTERS, VOL. 79
ISSN: 0065-2458/DOI: 10.1016/S0065-2458(10)79002-1

1. Introduction 72
1.1. The Missing Parallel Computation Model 76
1.2. The Portability Problem 78
1.3. The Scalability Challenge 81
1.4. A Simple Example—Portability and Scalability 83
2. Parallel Hardware Obstacles 85
2.1. The Multiarchitecture Problem 85
2.2. The Cache Coherence Problem 88
2.3. The False-Sharing Problem 91
2.4. A Simple Example—Cache Coherency and False Sharing 93
3. Parallel Software Issues 97
3.1. Data Dependency 98
3.2. Memory Consistency Model 102
3.3. Data Race Conditions 105
3.4. Locks and Deadlocks 107
4. The Human Factor 110
4.1. Teaching Parallel Programming Today 111
4.2. A Wish List 112
5. Conclusions 113
References 114

1. Introduction

> The device should be as simple as possible, that is, contain as few elements as possible. This can be achieved by never performing two operations simultaneously, if this would cause a significant increase in the number of elements required. The result will be that the device will work more reliably and the vacuum tubes can be driven to shorter times than otherwise.
>
> John von Neumann, 1945 [1]

The paradigm shift from sequential computing to parallel computing is a revolutionary leap, not an evolutionary step. It is a revolutionary leap because it changes almost any topic in the computer science discipline. The prefix parallel can be added to any topic: parallel architectures, parallel OS, parallel algorithms, parallel languages, parallel data structures, parallel databases, and so on. But the parallel

revolution is a revolution in progress: while advances in parallel hardware accelerate everyday, the development of parallel software lags behind.

Parallel computing is not a new concept. The first computers, such as the ENIAC [2] and IAS [3], were highly parallel machines and the lessons that were learnt from their design are still relevant today. The ENIAC was a parallel, data-flow machine in which program memory was locally stored in each unit. The IAS was an asynchronous machine with parallel memory and parallel arithmetic. However, although the computer pioneers—John von Neumann, Presper Eckert, John Mauchly, and Herman Goldstine—designed their computers to be highly parallel machines, they did not use the parallel features of the machines because of its complexity. They claimed that serial operation is preferred in electronic machines, provided the components are fast enough. As mentioned in their reports [1, 4, 5], the advantages of a serial machine over a parallel one are reduced hardware, increased reliability, and ease of programming. Therefore, parallel computing will remain a research niche for many years to come.

In the past two decades, parallel computing has remained an elitist discipline in that only professional programmers in rich scientific and engineering communities could afford the state-of-the-art parallel machines. Multiprocessor machines are very expensive and demand a highly specialized expertise in systems administration and programming skills. The turning point finally arrived in 2005, when parallel computing on any desktop became a reality [6]. Two things were missing prior to 2005: low-cost parallel computers and simple, easy-to-use parallel programming environments. However, they are available now and will change the way developers design and build software applications. The two complementary technologies bring parallel computing to the desktop. On the hardware side is the multicore processor for desktop computers; on the software side is the integration of the OpenMP parallel programming model [7, 8] into Microsoft Visual C++ 2005. These technologies promise massive exposure to parallel computing that nobody can ignore; a technology shift is therefore unavoidable. The HPC research community has made astonishing progress in this period of time. Today, efforts are focused on translating the knowledge of building off-chip supercomputing based on multiprocessors to on-chip hypercomputing based on multicore processors.

Two *walls* have forced chip makers to change processor design to multicore processor architectures: the performance-wall and the power-wall.

The *performance-wall* refers to the observation of a slowing down in performance improvement of single-core processors. It is becoming increasingly difficult for processor designers to continue to use pipelining and superscalar techniques to enhance the speed of modern processors (Fig. 1) [9].

The *power-wall* refers to the observation of an exponential increase in the power consumption of single-core processors due to an increase in the number of

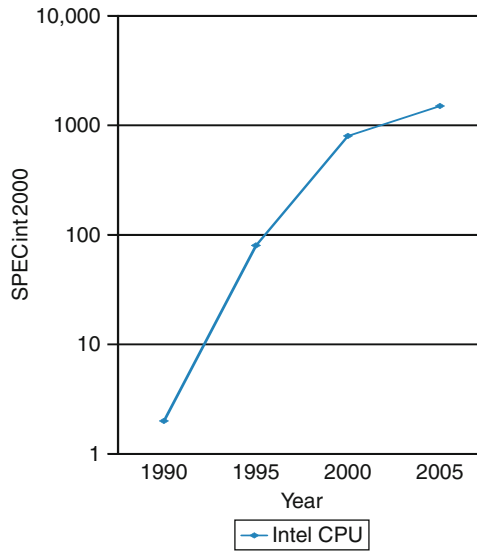

FIG. 1. The performance wall.

transistors on a single die for implementing pipelining and superscalar techniques in higher frequencies (Fig. 2) [9, 10]. Thus, replacing the large superscalar processor with several small cores with a lower clock rate achieves better performance per watt.

The primary consequence of the transition to multicore processors is that applications will increasingly need to be parallelized to fully exploit the throughput gains now becoming available. Unfortunately, parallel code is more complex to write than that of serial code [11, 12]. *Parallel programming is no doubt much more tedious and error-prone than serial programming, but it is not impossible*. The main difficulties in programming multicore processors are as follows:

– *Thinking in Parallel*. Parallel programming demands a different way of thinking. Sequential programming is a deterministic and linear process. Therefore, it is very intuitive to the way we solve problems through the use of algorithms. In contrast, parallel programming is a multiprocess approach where the behavior of the processes is intrinsically nondeterministic. Parallelism requires programmers to think in a way humans find difficult. However, allowing nondeterminism in parallel programs is the key to achieving high-performance and scalable programs. The nondeterminism in a parallel

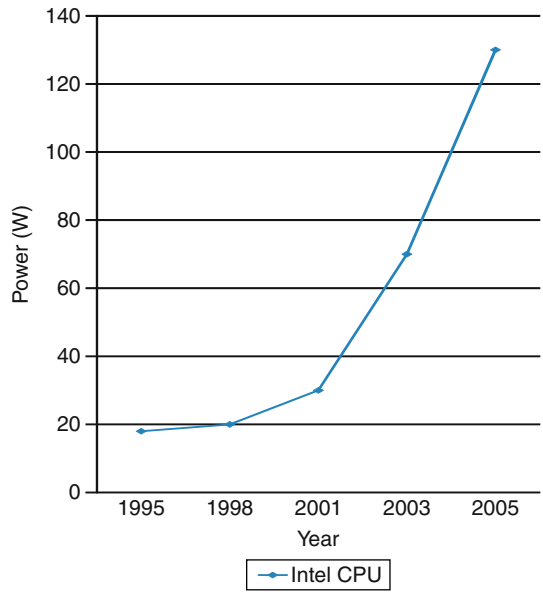

FIG. 2. The power wall.

program is controllable when it is necessary. Moreover, it is possible to learn how to think in parallel and how to write bug-free parallel programs [13].

– *Familiarity with Underlying Architecture*. Parallel programming demands a close familiarity with the details of the underlying multicore processor architecture. This knowledge is needed for matching the logical-structure of the program to the physical architecture of the processor. The programmer has to know such details as the number of cores, the memory layout, the cache memory hierarchy, and much more. Better matching leads to better performance and achieves desirable scalability. Sequential programming frees the programmer from this entire burden.
– *New Programming Issues*. Parallel programming demands the understanding of new programming issues that are absent from sequential programming, including: the relationship between the logical threads and the underlying physical processors and cores; how threads communicate and synchronize; how to measure performance in a parallel environment; and the sources of load unbalancing. The programmer must check for dependencies, deadlocks, conflicts, race conditions, and other issues related to parallel programming.

– *Invisible Problems*. Many parallel programming bugs are caused because they cannot be detected just from inspection of the source code. Actually, What-You-See (in the source code) is not What-You-Get (after optimizations). For example, aggressive optimizations such as branch prediction and reordering instructions for out-of-order execution are done ''under-the-hood'' and can change the meaning of programs. Moreover, these techniques are often the main cause of tricky data-race problems that are very hard to detect.
– *Difficult Debugging*. Debugging a parallel program is a highly tedious and difficult task. Although parallel debuggers and visual profiling analyzers exist and are improving all the time, finding a bug in a parallel program is like finding a needle in a haystack. The complexity of parallel debugging is due to the invisible problems and to the timing complexity of parallel program flow that harden on finding temporary bugs, whose appearance cannot be predicted.

The aim of this chapter is to explain the main difficulties and barriers to successful manycore programming. First, the unsolved problem of parallel computation model and its implications on issues such as portability and performance prediction is discussed. Next, the obstacles incurred by parallel hardware architectures on software development are covered. Then, the main traps and pitfalls of multicore programming are addressed. Finally, the human factor on the success of the parallel revolution is presented.

Terminology Comment: Throughout this chapter the term *parallel revolution* is used, and not the more popular term the *concurrency revolution*, because it better reflects the essence of current revolution. *Concurrency* is used in computer science to refer multiple software threads of execution that are running in an *interleaving mode* on a single-core processor, while, the *parallel* is used to refer multiple threads of execution that are running *simultaneously* on multicore processor.

1.1 The Missing Parallel Computation Model

The goal of a computation model is to establish an abstract and theoretical framework for describing and evaluating algorithms in a way that predicts their performance on real computers. Moreover, the main role of a computational model is to serve as a bridging model between the computer (the hardware) and the algorithm (the software) together with a simple cost function for performance prediction.

The Von Neumann model serves as a universal sequential computation model. The blossoming of the computing industry in the last six decades is evidence that the Von Neumann model is a success story as it offers a simple model of computation; it enables the design of simple and easy-to-use programming languages founded on it;

it is based on architecture independency; guaranteeing performance among a variety of sequential architectures; and it enables the foundation of a simple cost model.

Despite many efforts, no solid framework exists for developing parallel software that is portable and efficient across multiple processors. The lack of such a framework is mostly due to the absence of a universal model of parallel computation which can play a role similar to Von Neumanns model that plays for the sequential computing, inhibiting the diversity of the existing parallel architecture, and parallel programming models [14]. The lack of a unified parallel computation model, and therefore of a generic parallel programming model, is the main reason for the postponement of the shift of parallel computing to the mainstream.

Many parallel computational models, parallel programming models, and parallel languages have been introduced in the last three decades [15]. Parallel computational models such as PRAM and its extensions are pure theoretical models while others such as LogP [16] and BSP [17, 18] are practical models. The PRAM model is an attempt to follow the role of the RAM model in sequential computing. However, the PRAM model fails to introduce an adequate performance model that enables one to predict the performance of parallel programs on real parallel machines. On the other hand, parallel models such as LogP and BSP introduced a simple cost model based on four parameters that capture the complexity of the underlying architecture and improve the accuracy of performance prediction by computational analysis.

Unfortunately, there is no consensus in the HPC community on which model to choose as a standard. As a result, a variety of parallel machine architectures exists among different computer manufacturers or among multicore processor generations made by the same vendor. This situation leads to a very difficult portability problem where the rewriting of parallel programs, as each new machine or a new multicore processor appears, is unavoidable. In the last decade there were unsuccessful attempts to combine the three main parallel programming paradigms (data parallel, message-passing, and shared-variable) into a unified framework.

As philosopher of science, Thomas Kuhn explains in his book, *The Structure of Scientific Revolutions* [19], the acceptance or rejection of a scientific revolution is more influenced by social conditions than by scientific facts. According to Kuhn's observation, a *paradigm shift* (a term coined by Kuhn) occurs after three phases. The first phase, the *preparadigm phase*, in which there is no consensus on any particular theory; the second phase, *normal science*, in which there is a widespread consensus on the appropriate research directions that the community has to follow; and the third phase, *revolutionary science*, a period of crisis where anomalies in research reveal the weakness of a paradigm. At the end of the crisis a new paradigm is accepted. The *preparadigm phase* of the parallel computing revolution was the period of the first three decades of the sequential computing era, when there was no consensus on parallel computing feasibility and practicality. The *normal science*

phase started two decades ago, when the scientific community began to explore parallel machines and programming paradigms. The third phase started in the year 2005, when the first desktop multicore processors appeared, marking the beginning of the crisis period, the *revolutionary science*.

The efforts of manycore research to make parallel programming a general-purpose programming are concentrated today in four research centers. The Universal Parallel Computing Research Center (UPCRC) at University of Illinois at Urbana-Champaign planned a research agenda that ''aims to make client parallel programming synonymous with programming...to develop disciplined parallel programming models and languages, supported by sophisticated development and execution environments, that will offer the analog of modern sequential programming principles of safety, structure, and separation of concerns'' [20].

The Pervasive Parallelism Laboratory (PPL) at Stanford University focuses on ''making parallel programming practical for the masses'' by year 2012 [21]. The research aims to create a unified platform (algorithms, programming models, runtime systems, and architectures for scalable parallelism) that supports 10,000 hardware threads.

The Parallel Computing Laboratory at Berkeley sees the ''ultimate goal as the ability to productively create efficient and correct software that scales smoothly as the number of cores per chip doubles biennially'' [22]. Key to this approach is a layer of libraries and programming frameworks centered on the 13 computational bottlenecks (''dwarfs'') that they identified in the original Berkeley View report [23].

The DARPA High Productivity Computing Systems (HPCS) program is focused on providing a new set of tools that improve programmability, portability, robustness, and productivity of HPC applications [24]. The HPCS program is developing three parallel programming languages (Chapel, Fortress, and X10) focused on improving programmer productivity. These languages are looking for novel approaches to express computation by mathematical notation (Fortress); new parallel programming concepts based on object-oriented paradigm (X10); and new high-level abstractions for data, task, and nested parallelism.

1.2 The Portability Problem

Software portability, or the degree to which application can be moved from one environment to another and get essentially the same results, is obviously of supreme practical importance.

Performance is often cited as the main reason for the current push toward manycore processors. Performance is hard to achieve and requires programming time and programmer expertise. Expertise is necessary to understand the characteristics of the

machine and the application. Time is necessary to find the best ways to exploit them by transforming and specializing the code.

Portability across multicore processors and operating systems also requires programming time and programming expertise. It may be easier to achieve this with scientific applications rather than with interactive software, but it always requires code generalization. Portability is attained by ignoring the target architecture as much as possible.

Parallel *Performance Portability* is much more difficult than sequential portability. The lack of parallel software portability is mainly because high-speed computers support parallelism in many different forms, such as instruction-level parallelism, vector parallelism, loop-level parallelism, and task parallelism. Furthermore, data can be placed in many different types of storage, such as registers, caches, local memories, or shared memories. Today's multicore architectures are very diverse in the levels of parallelism and data storage they provide. If one has to find the best mix of these resources for every program then the lack of program portability is unavoidable. However, even if the management of these resources is provided by sophisticated hardware mechanisms, compilers, or operating systems, the programmer often has to take special care in writing the programs so that these mechanisms can be effective. This means, for example, that one must make sure that data placed in longer latency storage by the compiler get accessed infrequently, or that the levels of parallelism supported automatically by the given machine will be apparent to the optimization tool.

There is a need for efficient matching techniques to cope with the parallel portability problem caused by the diversity of parallel architecture and parallel programming models. Matching complex parallel applications to complex manycore processors in order to exploit the efficient use of available resources is the main challenge toward achieving *scalable portability* [25–28]. Parallel applications induce different logical structures, such as tree, mesh, hypercube, and butterfly. In the past two decades the motivation of the vendors of parallel computers to build machines based on these topologies stems from the ambition to match the physical topologies to the geometry and logical structures of common algorithms in computer science. For example, fluid dynamic problems often induce 2D/3D dimensional grid structures. Divide-and-conquer algorithms induce tree structures; the FFT algorithm induces a butterfly structure; and the Batchers sort algorithm induces a hypercube structure.

The increasing complexity of parallel applications and parallel machine architecture demands a higher level of parallel programming models, languages, and tools that hide these complexities from the programmer. To preserve the simplicity, ease of use, and portability of the programming workmanship, there is a need to move up to a higher level of programming abstractions. Many effective techniques have been developed in the last decade and have drastically reduced the transfer cost within

certain areas. Each has its own relative merits; some methods are cheap, others efficient; while still others are wide ranging [25, 29–31]. These techniques are now making a comeback [22, 23].

For example, *Auto-tuners* [29, 30, 32] are high-level libraries that use automatically calibrated statistical models to select from among several implementations and to optimize the parameters of each implementation. Model-based parameter optimization provides a form of adaptive algorithm. This leads to portability with high-performance because the library selects and tunes the best implementation for the current platform and workload.

The *Template*/*Skeleton*/*Sketching* [33, 34] is another concept to achieve portability. This approach restricts the computational model seen by programmers and allows them to use only a small set of skeletons or high-level parallel language constructs. The main advantage of this approach is the possibility of devising automatic or quasi-automatic tools that guarantee portability and performance of the code produced. Parallel constructs correspond to those forms of parallelism (pipeline, task farm, data parallel, and reduce) that are more often encountered in parallel algorithms. For each target architecture, an implementation template can be associated with each of these forms of parallelism. Typically, for a given distributed memory MIMD machine, a template consists of a set of communicating processes with mapping directives on the processor network, and with an analytical performance model used to automatically tune parameters such as degree of parallelism and task granularity. The template concept seems to hold some promise in addressing the problem of portable expression of parallel algorithms and applications.

A software component can be developed as a generic module to increase its degree of portability, parameterized by types, by operations, or by other modules. Since different architectures have different performance-relevant features, a portable program must execute different sequences of source-code instructions on different machines. This requires having different *program variants*, the selection of which is determined by the programs *tuning parameters*. A good generic model allows these tuning parameters to be calculated from its machine parameters. The drawback of this approach is that generic, but suboptimal, methods must be chosen. This results in a more complex source code with an attendant decrease in performance. Rewriting an algorithm for a refined data-type can be automated if the relationship between the abstract and realization data-types is specified accurately by an abstraction function. Consequently, it is possible to generate several versions of an algorithm for different representations of the data, after which these algorithms can be compared with respect to performance.

Similarly, it is possible to generate several executables, each implementing different algorithms and data-structures based on the special characteristics of the computer under consideration. If automatic generation of source-code is a goal, all

development steps must be worked out precisely. This may be tedious, depending on the tools that are available.

A necessary, but still not sufficient, condition for a parallel programming model to become a portable one is its widespread adoption by the community. For example, the OpenMP [7, 8] parallel programming model has been adopted by the entire software community and can be found as an integral part of any compiler and operating system. This is the reason for choosing OpenMP for the demonstration examples in this chapter. However, OpenMP cannot preserve this achievement because of the lack of a cost model as well as deviations from the formal specification, such as extended features which are not part of the standard, allowing behaviors called *implementation defined* to vary among different compliant implementations. However, an aware programmer can avoid using features that might cause unspecified behaviors.

1.3 The Scalability Challenge

Scalability of a parallel system is the ability to achieve more performance as processing nodes increase. A system (hardware + software) whose performance improves after adding more nodes, proportionally to the number of nodes added, is said to be a *scalable system*. Obtaining a scalable manycore processor system is the most challenging issue of the parallel revolution.

It is worth noting if the underlying manycore processor architecture is scalable but the software is not, and vice versa. Both the hardware and the software have to exhibit a *combined* scalable system. Intuitively, a scalable system has to be able to *scale up*. However, future manycore-based systems will also have to exhibit the ability to *scale down* to reduce costs, save power consumption, and to enable modern parallel multitasking operating systems, based on techniques such as *gang-scheduling* [35], to schedule a parallel application to a different number of cores for better system utilization.

With regard to hardware, the main scalability challenge of the designers of manycore architectures is to cope with the *bandwidth wall* problem. The bandwidth wall refers to the situation in which bandwidth limitation becomes a bottleneck that limits 4–64 core processors (multicore) to scale up to 100–1000 core processors (manycore).

Over the last two decades, memory performance has been steadily losing ground to CPU performance. During this period, CPU speed improved at an annual rate of 55%, while memory speed only improved at 10%, causing the CPU-memory performance gap to widen. Therefore, the bandwidth and latency of main memory (static DRAM) have not kept pace with CPU performance. Currently, processor caches provide access to data with latencies 100 times less than DRAM latencies. Bandwidth, likewise, is reduced as one move from cache to main memory.

Higher aggregate bandwidth and reduced latency are needed for core-to-memory high-volume traffic and for core-to-core communications. It is not improbable to assume that off-chip communications interconnects such as crossbars, meshes, and trees will be implemented on-chip aside from other new communication technologies that will be developed. However, network scalability is limited not only by topological properties such as node degree, diameters, symmetry, and bisection-bandwidth, but also by physical technology and management requirements such as packaging, power dissipation, and cooling.

Over the last two decades, the chipmakers have coped with the bandwidth wall problem by increasing the cache memory size (from KB to MB) and by introducing sophisticated cache techniques (smarter cache hierarchies). However, it seems that manycore processors will need new approaches. Two promising technologies have the potential to reduce the CPU-memory gap: 3D memories devices and optical interconnects. HP Labs call for on-chip optics and argue that a 256-core will deliver as much as 10 Teraflops of performance and will need as much as 20 TFlops bidirectional bandwidth to support a relatively flat programming model [36].

Intel lists five challenges for IC scaling [37] and claims that chip-to-chip optical interconnects can address the bandwidth bottleneck if future technologies will find a way to effectively integrate photonics with silicon logic. Using optical interconnects for on-chip signaling may be further off in the future due to the difficulties of scaling optical transceivers and interconnects to the dimensions required. Future architectures will require bandwidths of 200 GB/s–1.0 TB/s for manycore tera-scale computing [38].

With regard to software, the enemy of scalability is serial code. Amdahls Law is based on the observation that parallel portions of a program can be sped up infinitely in theory, but the serial portion is always serial and will always limit the speed of parallel code. Serialization of parallel code can be caused by legitimate serial code, such as I/O operations, but also by unnecessary overhead. Such overhead is incurred by using a *mutual exclusion* mechanism, such as looks, for synchronizing simultaneous access to shared memory by multiple threads. Unfortunately, lock-based synchronization has its downside. Coarse-grained locking does not scale well, while the more sophisticated fine-grained locking exposes the code to deadlocks and data races. Therefore lock-free programming paradigms attract many researchers. One of the emerging lock-free programming concepts is *transactional memory* (TM) [39, 40]. However, TM programming is still in the research stage and does not resolve all parallel programming challenges. Moreover, there are practitioners who claim that writing scalable and correct multithreaded code with locks is possible and offer a bag of tricks [41]. A recent research supports this claim and shows that state-of-the-art parallel programming models such as the Intel Threading Building Blocks (TBB) are able to deliver fine-grained locking while freeing the programmer from dealing with locks [42].

1.4 A Simple Example—Portability and Scalability

The following example demonstrates the scalability and the performance portability issues of contemporary multicore processors. The experiments were conducted on a laptop computer equipped by Intel Core 2 Duo CPU T8100, 2.1 GHz, and 1 GB of RAM. On the software side, the compilers used were MS Visual Studio C++ 2005 and Intel C++ version 11.0 on top of the MS Windows XP Professional operating system. The compiler options were set to their default values. The parallelization was done by OpenMP 2.0. The results shown have an average time of 20 runs of 10 measurements each.

Listing 1—Loop Parallelization Version 1.

```
float A[100000], B[100000];
void a1 (int n, float *a, float *b)
{
  int i ;
  #pragma omp parallel for
  for ( i=1; i<n ; i++) /* i is private by default */
    b[i] = (a[i] + a [i-1]) / 2.0;
}
```

The data-parallel example (listing 1) parallelizes a simple loop using the OpenMP compiler directive. This is the first example that appears in the OpenMP specification. Compiling the code with the MS Visual Studio C++ compiler and running it with two threads yields a speedup of 0.53 as compared to a run with one thread. There is nothing wrong with this code; the poor performance is due to the low work granularity inside the loop. The amount of work inside the loop does not compensate for the overhead incurred by the memory subsystem.

Now, let us put more work inside the loop. The modified code (listing 2) was run with a granularity of 1, 10,100, and 1000, and the measurements were taken for Microsoft and Intel compilers. The results are plotted in Figs. 3 and 4.

Listing 2—Loop Parallelization Version 2.

```
float A[100000], B[100000];
void a1(int n, float *a, float *b)
{
  int i, j, granularity ;
  #pragma omp parallel for
  for ( i=1; i<n ; i++) /* i is private by default */
    for (j=0; j < granularity ; j++)
      b[i] = (a[i] + a[i-1]) / 2.0;
}
```

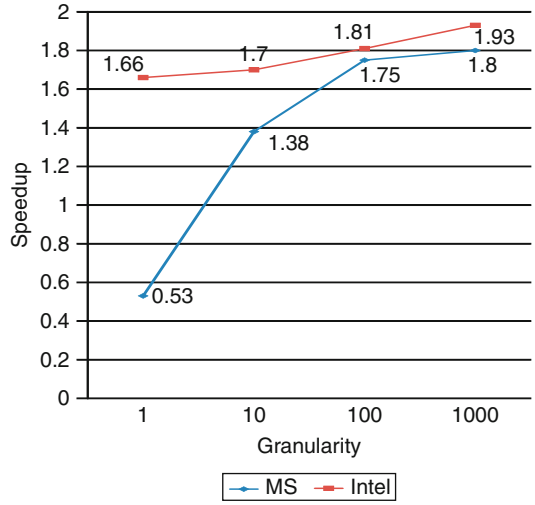

FIG. 3. Loop parallelization speedups for Microsoft and Intel compilers.

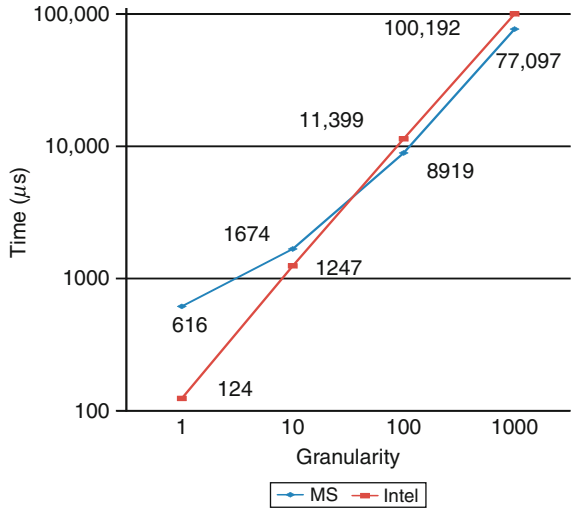

FIG. 4. Loop parallelization times for Microsoft and Intel compilers.

Figure 3 plots the speedup as a function of the granularity. First, it can be observed that the behaviors of the compilers are not the same. In case of Microsoft, the impact of the granularity on the speedup is more significant as compared to Intel. Second, the scalability of fine-grained computations is poor. Figure 4 plots the running times as a function of the granularity. It can be observed again that the two compilers achieve different performance gains. Intel outperforms Microsoft for fine-grained computations and the opposite is true for coarse-grained computations. The performance gap between the two compilers is up to a 500% difference for *only* two cores and two threads.

2. Parallel Hardware Obstacles

> I think today this shift toward parallelism is being forced not by somebody with a great idea, but because we do not know how to build hardware the conventional way anymore.
>
> David Patterson, 2007 [43]

> when we start talking about parallelism and ease of use of truly parallel computers, we are talking about a problem that is as hard as any that computer science has faced.
>
> John Hennessy, 2007 [43]

Designing and constructing manycore processors that scale up to hundreds of cores on a single die poses great challenges on the hardware architects. In the low level of the hardware, chipmakers will have to explore advanced technologies such as nonsilicon-based and nanoscale technologies. The discussion in this section concentrates on issues related to high-level hardware design aspects. First, the problem of multiarchitectures of multicore processors and its implications for software design is presented. Next, the cache coherence problem and false-sharing issues are described followed by a multiversion example that demonstrates their impact on the performance of a parallel application.

2.1 The Multiarchitecture Problem

The *Multiarchitecture Problem* refers to the situation where parallel machines vendors and multicore chipmakers design their products based on different architectures. The existence of variety of parallel architectures is due to the absence of a universal parallel computation model that can be used as a common guidelines framework for constructing parallel machines and multicore processors. Therefore,

it is very difficult to create parallel programs that are able to preserve performance portability and scalability on different parallel machines and multicore processors.

The core of any parallel architecture is the interconnection network that links the system's nodes. During the last two decades, multiprocessor machines have been built with many different network topologies [44]. There are static topologies (2-2.5-3D Meshes, Hypercube, Tree, Ring, and shuffle-Exchange, among others) and dynamic topologies (Bus, Crossbar, Multistage-network, and Fat-tree, among others) and each topology has different properties (diameter, bisection width, edge length, and edges per node). Therefore, designing a parallel algorithm must take into account all these underlying hardware characteristics for efficient matching between the logical structure induced by the algorithm and the physical layout of the hardware. For example, reduction is a basic operation in parallel programming for the computation of summation, minimum, or maximum of a list of values. The reduction operation induces a tree structure that matches perfectly to a tree topology. But for other topologies, such as Mesh or Hypercube, efficient embedding of a tree structure into these topologies is required. Unfortunately, compilers do not do that automatically. Therefore, the matching task is left to the programmer who has to do it manually.

Embedding of an algorithmic structure into a network topology can help but does not solve the problem. Usually, the logical layout of an algorithm is more complex and the only way to achieve scalable performance is to design different algorithms *that solve the same problem* for different topology networks. Table I lists the complexities and the *Isoefficiency* functions of two practical algorithms, the *bitonic sort* algorithm and *Dijkstra's Single-Source Shortest Paths* algorithm, on three different topologies [45]. The isoefficiency function determines the degree of

TABLE I
THE COMPLEXITY AND SCALABILITY DEGREES OF BITONIC-SORT AND DIJKSTRA'S ALGORITHMS FOR HYPERCUBE, MESH, AND RING TOPOLOGIES

	Algorithm		
	Bitonic sort		
Architecture	Maximum processors	Parallel run time	Isoefficiency function
Hypercube	$\Theta(2^{\sqrt{log(n)}})$	$\Theta(n/(2^{\sqrt{\log(n)}})\log(n))$	$\Theta(p^{log(p)}log^2(p))$
Mesh	$\Theta(log^2(n))$	$\Theta(n/log(n))$	$\Theta((2^{\sqrt{p}})\sqrt{p})$
Ring	$\Theta(log(n))$	$\Theta(n)$	$\Theta((2^p)p)$
	Dijkstra's single-source shortest paths		
Hypercube	$\Theta(n/log(n))$	$\Theta(nlog(n))$	$\Theta(p^2log^2p)$
Mesh	$\Theta(n^{0.66})$	$\Theta(n^{1.33})$	$\Theta(p^3)$
Ring	$\Theta(\sqrt{n})$	$\Theta(n^{1.5})$	$\Theta(p^4)$

scalability of the parallel system. A small isoefficiency function indicates that the parallel system is highly scalable. The maximum number of processors indicates how far the system can scale up while maintaining a cost-optimal formulation. The data of Table I confirms that the impact of a network's topology, the number of processors, and other characteristics on the performance and scalability of a parallel algorithm is significant. Now it is clearer why the motivation of current research efforts is to develop *auto-tuners* and *sketching* techniques [32].

Parallel architectures do not differ from each other only by the network topology. They are also designed based on different concepts. There are parallel vector machines versus processor-arrays architectures; shared-memory and distributed-memory architectures; Single Instruction Multiple Data (SIMD) architectures versus Multiple Instructions Multiple Data (MIMD) architectures; Asymmetrical Multi-computers versus Symmetrical Multicomputers architectures; Clusters architectures and hybrid architectures [46].

Contemporary multicore processors contain two to eight cores. Currently, most of them do not contain interconnection networks (there are exceptions such as SUN Niagara processors that use Crossbar and Tilera 64 that uses 2-D Mesh) but it is most likely that future manycore processors will use some kind of interconnection networks on-chip. However, the first generations of multicore processors architectures also differ from each other in many other aspects. There are architectural differences among different processors vendors and also among multicore processors generations from the same vendor.

Figures 5 and 6 describe the layouts of the first Intel multicore processors. It can be observed that the main architectural differences among the processors lie in their cache-memory subsystems. The first Intel multicore processor, the Pentium D, is a dual-core processor and contains ''distributed'' L1 and L2 cache-memories; the next processor generation. The Core 2 Duo is a dual-core processor with separated L1 cache-memories and shared L2 cache-memory. The third generation, the Core 2 Quad, is a quad-core processor containing two separated ''Core 2 Duo'' processors. The last generation, the Corei7, is a quad-core processor with separated L1 and L2 cache-memories and shared L3 cache-memory. Some of the processors were launched with Simultaneous Multithreading (SMT) technology (or Hyperthreading as it is called by Intel) while others have not included that technology.

There are other architectural features of Intel processors (and other vendors as well) that are changed from one processor generation to another one, such as cache associativity, occupation policy (Intel usually uses inclusive policy while AMD uses exclusive policy), and cache coherence protocols. However, the point is clear. Each one of these processors demands a specific programming treatment for gaining the expected performance and scalability [47]. For example, Intel recommends a Client–Server programming strategy with Intel Core 2 Duo processor [48].

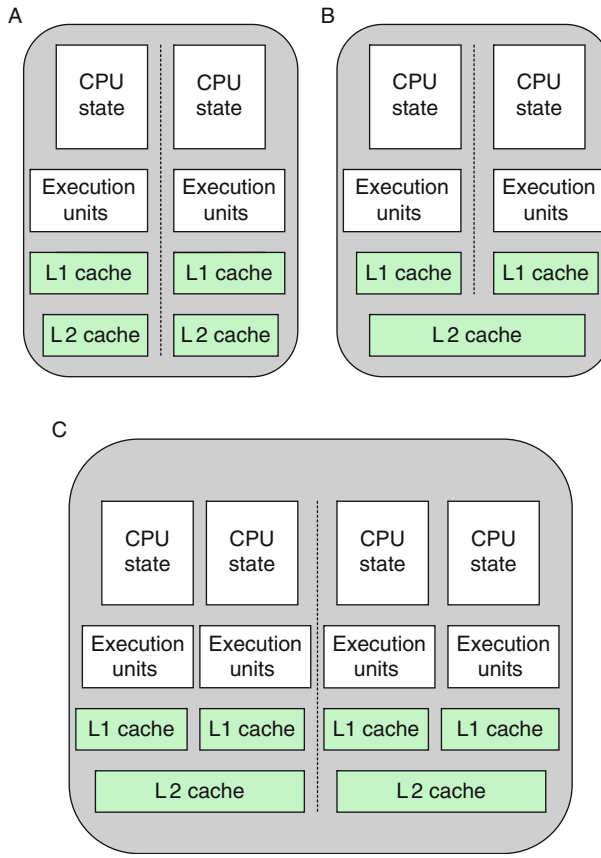

FIG. 5. Intel Multicore Generations: (A) Pentium D (B) Core 2 Duo (C) Core 2 Quad.

The reason is that the shared L2 cache of the Core 2 Duo is not banked. The L2 cache serves only one core at any given clock cycle while a round robin scheme is used to allocate L2 cache services to the cores for instances where both cores request L2 service. Therefore, many applications do not scale well on multicore processors [49].

2.2 The Cache Coherence Problem

Maintaining the coherence property of a multilevel cache-memory hierarchy (Figs. 5 and 6) incurs another serious performance problem known as the *cache coherence problem*. An inconsistent memory view of a shared piece of data might

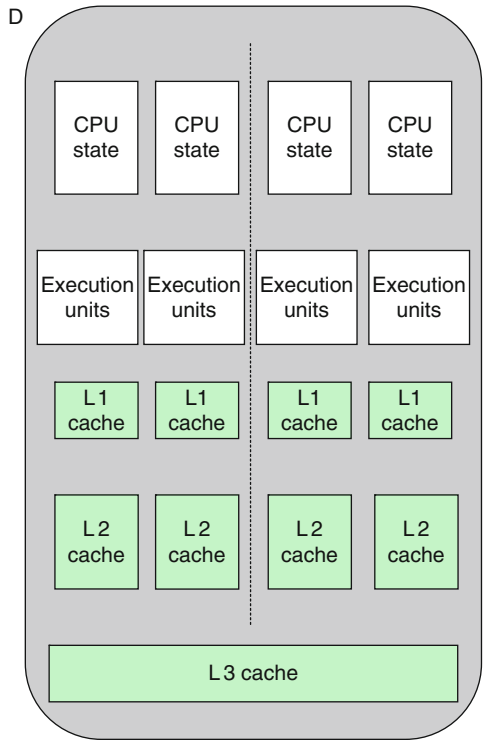

FIG. 6. Intel Multicore Generations: (D) Corei7.

occur when multiple caches are storing copies of that data item. In such a case, the memory subsystem is acting to insure a coherent memory view for all cores. Figure 7 illustrates the coherency problem. Two different cores, A and B, read the value of a shared variable V (Fig. 7A and B). Then, core B writes a new value to variable V (Fig. 7C). At this point, core A has an obsolete value of variable V (without knowing that the value is not valid anymore). Therefore, immediate action is required to invalidate the copy of variable V stored in the cache of core A. Otherwise, the inconsistent view of the main memory will lead to unexpected results.

Multicore processors apply coherency control, called *Snoopy Coherency Protocols*, in the writing of shared data. Two classes of protocols are most common: *write-invalidate* and *write-update*. The *write-invalidate* protocol invalidates all cached copies of a variable before writing its new value. The *write-update* protocol broadcasts the newly cached copy to update all other cached copies with the same value.

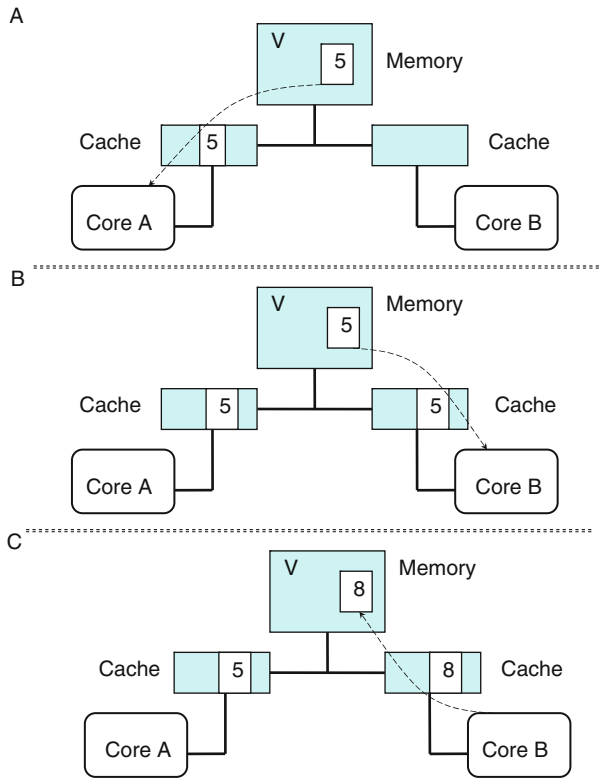

FIG. 7. Illustration of the cache-coherence problem.

The underlying mechanisms that implement these protocols constantly monitor the caching events across the bus between the cores and the memory subsystem and act accordingly, hence the term *snoopy* protocols.

The memory-subsystems architectures of multicore processors are designed as bus-based systems that support Uniform Memory Access (UMA). However, it is probable that memory-subsystems architectures of manycore processors will be network-based that support Nonuniform Memory Access (NUMA). Since snoopy coherency protocols are bus-based protocols, they are not appropriate for scalable NUMA-based architectures. For applying coherency control in NUMA systems, *Directory-Based Coherency Protocols* are usually used. The directory approach is to use a single directory to record sharing information about every cached data item. The protocol constantly keeps track of which cores have copies of any cached data

TABLE II
LATENCY COSTS OF CACHE-MEMORIES ON AN INTEL CORE DUO SYSTEM

Case	Data location	Latency (cycles/ns)
L1 to L1 Cache	L1 Cache	14 core cycles + 5.5 bus cycles
Through L2 Cache	L2 Cache	14 core cycles
Through Memory	Main Memory	14 core cycles + 5.5 bus cycles +40–80 ns

item, so that when a write operation is invoked to a particular data item, its copies can be invalidated.

Coherency protocols incur substantial overhead that inhibits scaling up the multicore systems. An on-chip deeper multilevel cache-memory hierarchy makes the cache coherence problem harder. Table II lists the latency costs that an application can suffer in loading data on an Intel Core Duo system [48]. As can be observed, a deeper cache-memory hierarchy leads to higher latencies. Now it is clearer what the obstacle is and where it lies to inhibit better speedup in the example of Section 1.4 (Fig. 3).

Cache-memory coherency management is a programmer-transparent obstacle that can be resolved by eliminating sharing data. Unfortunately, eliminating sharing data might lead to poor performance and nonscalable applications. Therefore, any solution must be carefully checked from all angles before final implementation. The example in Section 2.4 demonstrates the cost of the cache-memory coherency management.

2.3 The False-Sharing Problem

The cache coherent problem occurs when multiple cores are writing into a shared data item. Unfortunately, cache incoherency might also occur when multiple cores are writing into different data items that are located adjacent to each other, hence the name *false sharing*.

Figure 8 illustrates a false-sharing situation. Data items are moved between main memory and cache in chunks called *cache lines*, typically of size 64 or 128 bytes. If two cores are writing into different adjacent memory locations that are mapped to the same cache-line they may continuously invalidate each other's caches. Cores A and B in Fig. 8 read from and write into different elements of array V. Core A reads from and writes into the first element of array V, while core B reads from and writes into the second element of array V. After both cores read from V, each one stores the same cache-line in its private cache. When core A updates its element to the value 1,

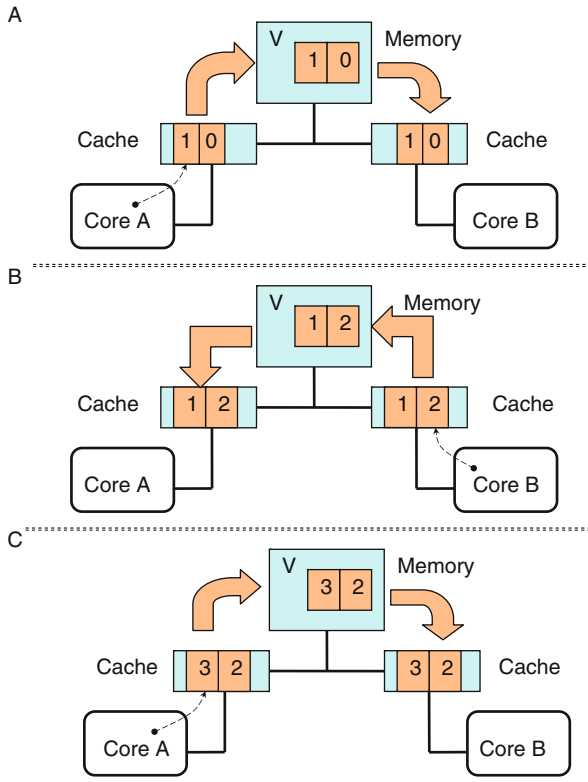

FIG. 8. Illustration of the false-sharing problem.

the shared cache-line must be shipped to Core B's private cache (sometimes via the main-memory) although the second element of array V remains untouched (Fig. 8A). Now, core B updates its element to the value 2, and the shared cache-line must be shipped to core A's private cache although the first element of array V remains untouched (Fig. 8B). And once again core A updates its element to the value 3, and again the shared cache-line must be shipped to core B's private cache (Fig. 8C).

False-sharing situations cannot be detected easily by inspecting the source code or by using profiling tools. Fortunately, there are simple rules and programming techniques that can be enforced for minimizing the occurrence of false-sharing situations. Such programming techniques include padding data structures with dummy data, aligning them in memory on cache-line boundaries, or keeping them

apart by allocating different threads to them. The example in the next section demonstrates such techniques.

2.4 A Simple Example—Cache Coherency and False Sharing

The following example demonstrates the overhead costs of different basic synchronization mechanisms and the impact of cache coherency management and false-sharing situations on the performance. The example program scans an array of 100M integers and counts the odd numbers. The experiments were conducted on a laptop computer equipped by Intel Core 2 Duo CPU T8100, 2.1 GHz, and 1 GB of RAM. For software, the compiler used was Intel C++ version 11.0 on top of MS Windows XP Professional operating system. The compiler options were set to their default values. The parallelization was done by OpenMP 2.0. The results shown are an average time of 10 measurements.

Listings 3–9 present different versions of the same program using different synchronization facilities and programming techniques. Figure 9 plots the running

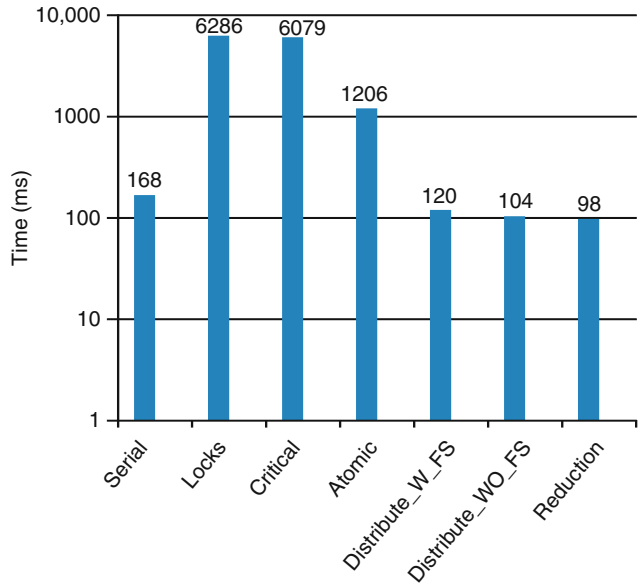

FIG. 9. Overheads costs of different synchronization mechanisms and programming techniques.

time results. Listing 3 presents the serial version of a counting program that lasts 168 ms. Listing 4 presents the first OpenMP parallel version of the counting program that uses *locks* for protecting the shared counter *count*. In this case, the program runs 6286 ms which exhibit the very high overhead incurred by the parallelism using locks. The program version using *critical-section* for protecting the shared counter (Listing 5) does not achieve much better result (6079 ms). Replacing the OpenMP *critical* directive with *atomic* directive (Listing 6) improves the performance dramatically (1206 ms). Now that the synchronization overhead is reduced to a minimum, the remaining overhead is mainly due to the cache coherency management.

Listing 3—Serial Counting of odd numbers.

```
#define N 100000000
int a[N], count=0;
void serial ()
{
  int i ;
  for (i=0; i<N; i++)
    if (a[i] % 2){
      count++; // count odd numbers
    }
  }
}
```

The impact of the cache coherency management is reduced by using distributed counters (Listing 7). Each thread employs a private counter (counter [0] and counter [1]) and their values are summed at the end of the counting. Now the total counting takes 120 ms, which is a speedup of 1.4. The private counters are located adjacent to each other and thus liable to incur false-sharing situations. To eliminate these false-sharing situations, the counters are padded by dummy data that prevents them from sharing the same cache-line (Listing 8). The running time after eliminating a false-sharing situation is 104 ms, an improvement of 13.3%, thus increasing the speedup to 1.61. The last experiment uses the OpenMP *reduction* operation (Listing 9). In this case, OpenMP implicitly creates private copies of the *count* variable for each thread. When the counting is over, the values of the private counters are added into the global variable *count*. By using the OpenMP reduction operation, the running time achieved is 98 ms, a speedup of 1.71. Therefore, it is recommended to use ready-made parallel constructs and parallel libraries, where possible, because they are optimized for the target machines.

Listing 4—Shared counter protected by locks.

```
#define N 100000000
int a[N], count=0;
int num_thrds = 2;
void locks (int num_thrds)
{
  int i ;
  omp_lock_t lck ;
  omp_init_lock(&lck) ;
  #pragma omp parallel for num_threads (num_thrds)
  for (i = 0; i<N; i++) /* i is private by default */
    if (a[i] % 2){
      omp_set_lock(&lck) ;
      count++;
      omp_set_lock(&lck);
    }
  }
}
```

Listing 5—Shared counter protected by critical section.

```
#define N 100000000
int a[N], count=0;
int num_thrds = 2;
void critical (int num_thrds)
{
  int i ;
  #pragma omp parallel for num_threads(num_thrds)
  for (i = 0; i<N; i++) /* i is private by default */
    if (a[i] % 2){
      #pragma omp critical
      count++;
    }
  }
}
```

Listing 6—Shared counter protected by atomic operation.

```
#define N 100000000
int a[N], count=0;
int num_thrds = 2;
```

```
void atomic (int num_thrds)
{
  int i ;
  #pragma omp parallel for num_threads(num_thrds)
  for (i = 0; i<N; i++) /* i is private by default */
    if (a[i] % 2){
      #pragma omp atomic
      count++;
    }
  }
}
```

Listing 7—Distributed counters.

```
#define N 100000000
int a[N], count=0, counter [2] ;
int num_thrds = 2;
void distributed_counters(int num_thrds)
{
  int i ;
  #pragma omp parallel num_threads(num_thrds)
  {
    int id=omp_get_thread_num () ;
    #pragma omp for
    for (i = 0; i<N; i++) /* i is private by default */
      if (a[i] % 2) {counter[id]++;}
  }
  count = counter[0]+counter[1] ;
}
```

Listing 8—Distributed counters with False-Sharing Elimination.

```
#define N 100000000
struct padding_int
{
  int value ;
  char dummy [60];
} counters [2];
int a[N], count=0;
int num_thrds = 2;
void distributed_counters_without_fs(int num_thrds)
```

```
{
  int i ;
  #pragma omp parallel num_threads(num_thrds)
  {
    int id=omp_get_thread_num () ;
    #pragma omp for
    for (i = 0; i<N; i++) /* i is private by default */
      if (a [i] % 2) {counters [id]. value++;}
  }
  count = count [0]. value+count [1]. value ;
}
```

Listing 9—Shared counter protected by reduction operation.

```
#define N 100000000
int a[N], count=0;
int num_thrds = 2;
void reduction (int num_thrds)
{
  int i ;
  #pragma omp parallel for reduction (+:count) num_threads
(num_thrds)
  for (i = 0; i<N; i++) /* i is private by default */
  if (a[i] % 2) {count++;}
  }
}
```

3. Parallel Software Issues

> I won't be surprised at all if the whole multithreading idea turns out to be a flop... I think it's a pipe dream... I also admit that I haven't got many bright ideas about what I wish hardware designers would provide instead of multicore, now that they've begun to hit a wall with respect to sequential computation.
>
> Donald E. Knuth, 2008 [50]

Manycore programming introduces new software programming issues that were absent from the sequential programming. This section presents the most bothersome and error-prone issues that the programmer has to cope with. These issues demand high programming skills by the programmer and have a significant impact on the performance portability, predictability, programmability, and the correctness of parallel applications.

3.1 Data Dependency

Parallelizing compilers that convert a given sequential program into an equivalent parallel program in an automatic manner were research-intensive topic during the last three decades. Unfortunately, the performance gain that can be achieved by automatic parallelization of modern compilers is only 5–10% on average [51]. Automatic parallelization poses many challenges and requires sophisticated compilation techniques to identify which parts of the sequential program can be executed in parallel. The main challenge is to find dependences that limit parallelism and then to transform the code into an equivalent code that ensures effective utilization of parallelism. The transformed program must preserve the meaning of the original computation and produce the correct results with every input.

The problem of data dependence analysis is intractable (NP-complete) and is followed by aggressive transformation such as loop interchange, loop splitting, loop distribution, vectorization, and parallelization to achieve efficient results across various parallel architectures. Therefore, these transformations are not usually found in classic optimizers and most of the tedious work of finding data dependences inside the source code and transform them into parallelizable code remains on the shoulders of the programmer. Fortunately, there are tools, such as Intel Threading Checker [52] and Intel Parallel Composer parallel lint feature [53] that can help the programmer find data dependences in source code and to advice on possible transformations. The following are the main definitions and terminology related to dependency theory:

Data Dependence Definition [54]:

There is a *data dependence* from statement S1 to statement S2 (statement S2 depends on statement S1) if and only if

1. Both statements access the same memory location and at least one of them stores into it.
2. There is a feasible run-time execution path from S1 to S2.

There are three kinds of data dependences expressed in terms of read-write order of statements S1, S2:

- *True dependence*: S2 has a true dependence on S1 iff S2 reads a value written by S1.
- *Antidependence*: S2 has an antidependence on S1 iff S2 writes a value read by S1.
- *Output dependence*: S2 has an output dependence on S1 iff S2 writes a variable written by S1.

Statements S1 and S2 can be executed in parallel iff there are no True, Anti, or Output dependences between S1 and S2.

Since loops are computational-intensive piece of code they are the first candidates for parallelization. Therefore, dependences in loops are deserved for formulation, definition, and treatment of their own:

Loop Dependence Definition [54]:

There exists a dependence from statement S1 to statement S2 in a common nest of loops if and only if there exist two iteration vectors i and j for the nest, such that

1. $i < j$ or $i = j$ and there is a path from S1 to S2 in the body of the loop,
2. Statement S1 accesses memory location M on iteration i and statement S2 accesses location M on iteration j, and
3. One of these accesses is a write.

If a loop nest contains loop dependence it is called *loop-carried dependence* otherwise it is called *loop-independent dependence*. Bernsteins conditions [55] treat dependences in loop from the opposite viewpoint and state that two iterations I1 and I2 can be safely executed in parallel if

1. Iteration I1 does not write into a location that is read by iteration I2.
2. Iteration I2 does not write into a location that is read by iteration I1.
3. Iteration I1 does not write into a location that is also written into by iteration I2.

The following example illustrates loop-carried dependence:

Listing 10—An example of loop carried dependence.

```
for (i=0; i<N-1; i++) {
  a(i) = b(i) + 2; // s1
  b(i+1) = a(i) - 6; // s2
}
```

This loop exhibits the dependence between s1 and s2 because on every iteration the computed value of array-a is immediately used in s2. There is also dependence between s2 and s1 because on every loop-iteration, other than the first iteration, the value of array-b that is used is the one that was computed on the previous iteration. Identifying these dependences is difficult, since different loop iterations access different elements of the arrays.

The loop-carried dependence in the loop above prohibits parallelization without synchronization. Fortunately, there is a simple transformation, called *loop distribution*, which can be applied to the loop for creating new opportunities for parallelism as follows:

Listing 11—An example of loop distribution transformation.

```
#pragma omp parallel {
#pragma omp for
  for ( i=0; i< N-1; i++){
    a(i) = b(i) + 2; // s1
  }
#pragma omp for
  for ( i=0; i < N-1; i++){
    b(i+1) = a(i)-6;//s2
  }
}
```

The code above is parallelized by OpenMP after the two statements, s1 and s2, were distributed between two successive and separated loops that each one can be parallelized.

Let us examine other cases of dependences and transformation techniques. In the example below, the loop cannot be parallelized because the antidependence and output-dependence caused by the assignments to the variable *t*:

Listing 12—An example of antidependence and output-dependence.

```
for (i=0, i < N, i++){
  t = a(i); // s1
  a(i) = b(i); // s2
  b( i ) = t; // s3
}
```

Applying *privatization transformation* on the variable *t* does not change the meaning of the code, but enables to parallelize the code by OpenMP as follows:

Listing 13—An example of privatization transformation.

```
#pragma omp parallel for private (t)
for (i=0; i < N; i++){
  t = a(i); // s1
  a(i) = b(i); // s2
  b(i) = t; // s3
}
```

Since loop distribution transformation eliminates loop-carried dependences, it can be used to convert a sequential loop to multiple parallel loops. However, this transformation implicitly inserts a synchronization barrier between the separated loops and thus decreases the granularity of parallelism and creates extra communication and

synchronization overhead. To minimize these side effects of the loop distribution transformation, a technique called *loop fusion* is applied to recombine fine-grained loops into dependence-free coarse-grained loops as demonstrated below:

Listing 14—An example of loop fusion transformation—before transformation.

```
for (i=0; i < N; i++){
  a(i) = b(i) + 1; // s1
  c(i) = a(i) + c(i - 1); // s2
  d(i) = a( i ) + 8; // s3
}
```

In the example above, there is a carried dependence from s2 to itself. Applying loop distribution transformation creates three separate loops as follows:

Listing 15—An example of loop fusion transformation—after loop distribution.

```
for ( i=0; i < N; i++){
  a(i) = b(i) + 1; // s1
}
for (i=0; i < N; i++){
  c(i) = a(i) + c(i - 1); // s2
}
for ( i=0; i < N; i++){
  d(i) = a(i) + 8; // s3
}
```

The first and the third loops above carry no dependence and therefore can be made parallel. Moreover, it is possible to recombine, or fuse, these loops into a larger loop without sacrificing parallelism. The final transformation is illustrated below:

Listing 16—An example of loop fusion transformation—after loop fusion.

```
# pragma omp parallel for
for (i = 0; i < N; i++){
  a(i) = b(i) + 1; // s1
  d(i) = a(i) + 8; // s3
}
for (i=0; i < N; i++){
  c(i) = a(i) + c(i - 1); // s2
}
```

Although the examples and the transformation techniques presented above are very encouraging, there are many cases where dependences cannot be identified for certain because they rely on data that are changed dynamically or because there is a need for a priori information that is known to the user only. The following example

illustrates a situation where the question of dependency depends on the values of the array-b that can be changed frequently before entering the loop:

Listing 17—Loop carried dependence that cannot be transformed.

```
for (i=0; i<N; i++)
  a[i] = f(a[b[i]]);
```

3.2 Memory Consistency Model

Manycore systems support cache-coherency mechanisms (see Section 2.2) for maintaining a consistent view of shared-memory in multilevel cache-memory architectures. However, imposing cache-coherency alone does not guarantee complete memory consistency that insures correctness and predictability of the running results of a multithread application. The reason is that the order of the memory accesses of one thread (running in one core) may be differently observed by another thread (running in another core) in each execution of the program, leading to inconsistencies and unexpected results. To overcome this problem there is a need for a well-defined policy that determines the ordering rules of memory events. Such a policy is called a *memory consistency model*.

The following example illustrates how uncontrolled read/write events that are executed under-the-hood and are not visible to the programmer may lead to events ordering that yield an unexpected result:

Listing 18—An example of invisible interleaving scenario (1).

```
Global MinValue = 10; //  initialization
Thread-0 (Core 0)                Thread-1 (core 1)
MinValue = Min (MinValue, 0);    MinValue = Min( MinValue, 1);
```

The expected outcome from the above simple example is that the value of MinValue is 0 at the end of this tiny program no matter which thread invokes the Min function first. Unfortunately, there is a possibility that the order of read/write events inside the Min function will lead to a counterintuitive result:

Listing 19—An example of invisible interleaving scenario (2).

```
Global MinValue = 10; //  initialization
Thread-0 (Core 0)                Thread-1 ( core 1)
Reg1=MinValue                    ...
Reg2=0                           Reg1=MinValue
If (Reg2<Reg1) MinValue=Reg2     Reg2=1
...                              If (Reg2<Reg1) MinValue=Reg2
```

Due to relative timing and speed of the hardware cores, the order of the read/write events of the program happens to be as presented above. In such a case, the value of MinValue ends up containing the wrong value of 1 and not the expected correct minimum value of 0. Such a phenomenon is called a *data race condition*. Moreover, the situation is more frustrating in practice because it happens sporadically and thus makes the debugging of such a problem a very tedious task. To prevent such a scenario in advance, the intervention of the programmer is a must. One simple solution is to call the Min function from inside a *critical section* as in the following OpenMP example:

Listing 20—An OpenMP solution to the invisible interleaving scenario.

```
int MinValue = 10;
#pragma omp parallel shared (MinValue) num_threads (2)
{
  int id ;
  id = omp_get_thread_num ( ) ;
  #pragma omp critical
  MinValue = Min(MinValue, id) ;
}
```

The next example illustrates another common optimization scenario of modern compilers and/or parallel architecture that leads to a surprising result due to the absence of synchronization between simultaneous read/write events of two independent threads:

Listing 21—An example of reordering scenario (1).

```
Shared-A=shared-B=0; //  initialization
Thread-0 (Core 0)                Thread-1 (core 1)
1: local -2 = shared-A           3: local -1 = shared-B
2: shared-B = 1                  4: shared-A = 2
```

In the example given above, the program uses local variables local-1 and local-2 and shared variables shared-A and shared-B. The shared variables are initialized to 0. It may seem that it is impossible for the program end up with the result local2 == 2, local1 == 1. Intuitively, either statement 1 or statement 3 should be executed first. If statement 1 appears first, it should not be able to see the write at statement 4. If statement 3 appears first, it should not be able to see the write at statement 2. Surprisingly, it does not work because compilers and architectures are allowed to reorder the read/write operations of individual threads. The reordering is allowed if it does not change the execution correctness of the individual threads:

Listing 22—An example of reordering scenario (2).

```
Shared-A=shared-B=0; //  initialization
Thread-0 (Core 0)                Thread-1 (core 1)
1: shared-B = 1                  3: local-1 = shared-B
2: local-2 = shared-A            4: shared-A = 2
```

Now, it is easy to see how the result local-2 == 2 and local-1 == 1 might occur.

Leslie Lamport introduced a strict memory consistency model called *sequential consistency* [56]. The sequential consistency model allows interleaving between the read/write events of individual threads but requires that all individual threads have to preserve their read/write events in the order they appear in the program. The interleaving scenario of the first example above (Listing 18) leads to unexpected result but does not violate the rules of the sequential consistency model. However, the second example above (Listing 21) violates the rules of sequential consistency model. The sequential consistency restriction inhibits the compiler and the underlying hardware from performing aggressive optimizations. However, it does not free the programmer from the responsibility of keeping the code unambiguous by implying proper synchronizations between threads operations and by avoiding breaking of necessary data and control dependencies.

Manycore processor and compiler designers are eager to make the whole system run faster. To do so, they apply aggressive optimizations and use programming techniques such as buffering and pipelining that violate the restrictions of the sequential consistency model. Therefore, alternative memory consistency models that implement weaker ordering constraints have been defined and are called *relaxed memory consistency models* [57]. For example, the Intel IA-32 architecture supports very strong memory consistency model but not as strong as sequential consistency because it allows read and write operations to be executed out of order. Therefore, IA-32 is classified as relaxed memory consistency model. Akhter and Roberts [47] show and explain how and why IA-32 breaks Dekker's Algorithm [58] as follows:

Listing 23—Dekker's Algorithm (1).

```
x = 0; y = 0; //  initialization
Thread-0 (Core 0)          Thread-1 (core 1)
x = 1                      y = 1
reg1 = y                   reg2 = x
```

Under sequential consistency in the Dekker's Algorithm above either reg1 or reg2 is set to 0, but never both of them, no matter how the instructions of the two threads are interleaved. However, the IA-32 allows reordering the instructions as follow:

Listing 24—Dekker's Algorithm (2).

```
x=0; y=0; //  initialization
Thread-0 (Core 0)          Thread-1 (core 1)
reg1 = y                   reg2 = x
x = 1                      y = 1
```

Now, both registers might set to 0 and therefore break the semantic of the algorithm, that is, to guarantee that at any time at most one of reg1 or reg2 will be set to zero.

Standardization of formal specifications of memory semantics, from the viewpoint of the processor and the parallel programming model, is essential for providing portability of parallel systems and for reducing the complexity of developing parallel application. However, no matter how strong the memory consistency model is, the programmer has to be familiar with the memory operation restrictions of the underlying architecture, the implications of the compiler optimizations on the program correctness, and the semantic behavior of the synchronization mechanisms supported by the programming model.

3.3 Data Race Conditions

Multiple threads accessing shared data simultaneously may lead to a timing dependent error known as *data race condition*. Data races may be hidden in the code without interfering or harming the program execution until the moment when threads are scheduled in a scenario (the condition) that break the program execution. Moreover, usually data races cannot be detected just by inspecting the source code as illustrated in the previous section in the case of Min function. Even a simple operation such as y + = 2 is usually translated by the compiler to two low-level operations (tmp = y; y = tmp + 1;).

The following example demonstrates how different timing scenarios lead to different results:

Listing 25—Data race illustration (0).

```
a=b=x = 0; //  initialization
Thread-0 (Core 0)          Thread-1 (core 1)
a = x                      b = x
x = a + 1                  x = b + 2
```

The above is the original code as appeared to the programmer upon inspection of the source code. The *x* variable is shared by the two threads and is initialized to 0. If the two threads execute their instructions in lock-step, then it is impossible to determine the value of *x* at the end of the program because the outcome of

simultaneous write operations to a shared variable is undefined. But if a write to x is done only by one thread at a time, then there are a few scenarios:

Listing 26—Data race illustration (1).

```
a=b=x=0; //  initialization
Thread-0 (Core 0)        Thread-1 (core 1)
a = x
                           b = x
                           x = b + 2
x = a + 1
>>>>>>> x is equal to 1
```

Listing 27—Data race illustration (2).

```
a=b=x=0; //  initialization
Thread-0 (Core 0)        Thread-1 (core 1)
                           b = x
a = x
x = a + 1
                           x = b + 2
>>>>>>> x is equal to 2
```

Listing 28—Data race illustration (3).

```
a=b=x = 0; //  initialization
Thread-0 (Core 0)        Thread-1 (core 1)
a = x
x = a + 1
                           b = x
                           x = b + 2
>>>>>>>> x is equal to 3
```

The three scenarios illustrated above yield three different results. However, none of these scenarios violate the rules of sequential consistency model. Data race conditions are usually resolved by using a lock that prevents unwanted thread interleaving situations.

There are tools, such as the Intel Threading Checker, that may help to detect hidden data races. However, even the best tool cannot detect every data race condition because it is intermittent, nondeterministic, nonrepeatable, and timing dependent problem where the number of the schedule permutations of threads is huge.

3.4 Locks and Deadlocks

Manycore programming uses multiple threads to access shared-memory objects. To avoid unpredictable situations such as race conditions, *mutual exclusion* techniques are used to impose constraints on the order that threads access a shared-memory location. An exclusive access to a shared object can be guaranteed by using locks, also called *mutexes*. Listing 4 illustrates a simple example of using OpenMP locks for protecting simultaneous access to a shared variable (*count*).

There are many types of mutexes, each one used in different kinds of situations and each one incur different amount of overhead. There are low-level primitive locking mechanisms (*semaphores*, *condition-variables*, *and mutexes*) and high-level locking mechanisms (*recursive-mutexes*, *read-write mutexes*, *spinmutexes*, *queuing-mutexes*, *and monitors*). For example, OpenMP supports the locking constructs *lock–unlock, atomic, single, and critical*. *Lock–unlock and critical* constructs are used for protecting coarse-grained critical sections where atomic is applied to a single assignment statement for protecting a single shared variable. Figure 10 is a bar chart of the OpenMP locking construct overhead measured by running the EPCC microbenchmarks [59] with two threads on three different platforms (Intel Pentium D, Intel Core 2 Duo, and Intel 2 Quad) [49]. Analysis of the results leads to the conclusion that it costs less to use the critical directive than to use the *lock–unlock*

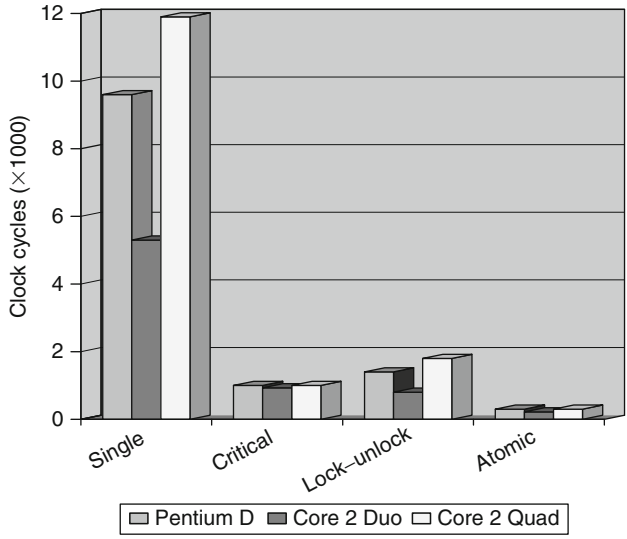

FIG. 10. OpenMP locking overheads of two threads on Intel Pentium D, Intel Core 2 Duo, and Intel Core 2 Quad machines.

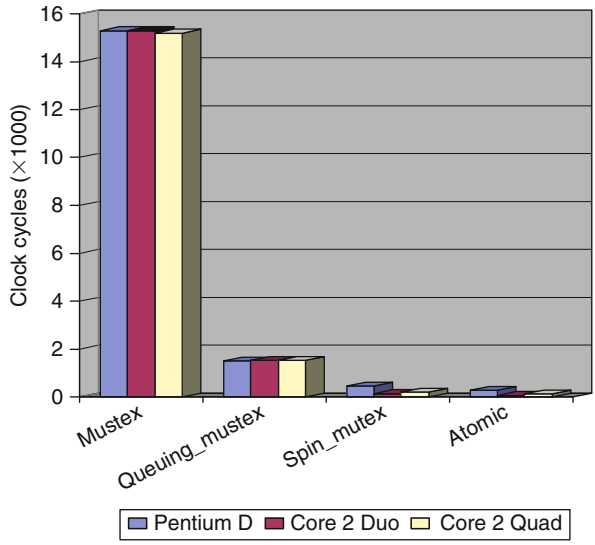

FIG. 11. TBB Locking overheads of two threads on Intel Pentium D, Intel Core 2 Duo, and Intel Core 2 Quad machines.

pair directive and since the overhead of the *atomic* directive is negligible is recommended for use, where possible, instead of the *critical or lock–unlock* directives.

Another example is the Intel TBB, which offers enhanced mutexes called *mutex*, *spin_mutex*, *queuing_mutex*, *and atomic*. For example, a task invoking a request to lock on *spin_mutex* waits (spins) until it can acquire the lock. It is very fast in low-contention scenarios and incurs very low overhead as shown in Fig. 11. *Queuing_mutex* is less desirable locking mechanism because it incurs more overhead though it offers fairness and avoids starvation by enforcing FIFO policy on the arriving locking requests [42].

Improper use of locks may cause many problems that are very difficult to detect without a sophisticated debugging tool. One such situation is known as *deadlock*. A deadlock is a situation in which a task A is waiting to acquire a lock on a shared object $r1$ locked by a task B, while locking a shared object $r2$ requested by task B. Since both tasks are blocked and waiting for release the object held by the other task, and none of them volunteers to be the first to release its object, the program execution is stuck. There are four conditions that lead to a deadlock situation:

– *Exclusiveness*: Exclusive assignment of an object to a task.
– *Multilock*: Allowing a task to acquire a lock on one object while locking another object.

– *Ownership*: A locked object can be released only by the task that holds it.
– *Cycling*: A task is willing to acquire a lock on an object held by another task that willing to acquire a lock on an object held by him.

Deadlocks can be avoided by breaking any one of these conditions.

Listing 29—An example of a deadlock caused by an incorrect locking hierarchy.

```
#include <stdio.h>
#include <omp.h>
int globalX = 0;
int globalY = 0;
int work0 ()
{
 omp_set_lock (&lck0) ;
  globalX++;
  omp_set_lock (&lck1);
   globalY++;
  omp_unset_lock (&lck1);
 omp_unset_lock (&lck0);
 return 0;
}
Int work1 ()
{
 omp_set_lock (&lck1);
  globalX++;
  omp_set_lock (&lck0) ;
   globalY++;
  omp_unset_lock (&lck0) ;
 omp_unset_lock (&lck1) ;
 return 0;
}
int main (int argc, char *argv [])
{
 omp_lock_t lck0 ;
  omp_lock_t lck1 ;
 omp_init_lock(&lck0) ;
 omp_init_lock(&lck1) ;
#pragma omp parallel sections
{
 #pragma omp section
```

```
    WORK0() ;
    #pragma omp section
    WORK1() ;
  }
    printf (''TOTAL = (\%d,\%d)\n'', globalX, globalY);
    omp_destroy_lock(&lck0) ;
    omp_destroy_lock(&lck1) ;
  }
```

The above example (Listing 29) illustrates the potential for deadlock because of an incorrect locking hierarchy. The two threads in this program invoke two functions (WORK0 and WORK1) that attempt to acquire two locks (lck0 and lck1) in reverse order for exclusive access of two global variables (globalX, globalY). If both threads obtain only the first critical section (an access to globalX) deadlock occurs because the second critical section (an access to globalY) never becomes available. The deadlock is avoided if one of the threads acquires both critical sections. This nondeterministic behavior of the threads execution can lead to situations where potential deadlocks lay dormant in the code for a long time without causing any damage until the day they suddenly appear for a moment and then disappear again. For example, Edward Lee reported on a case where a deadlock appeared 4 years after the application was launched [60].

One way to avoid deadlocks is to impose an ordering (eliminating cycles in the resource acquisition graph) on the locks and demand that all threads acquire their locks in the same order. Other techniques to prevent deadlock are *timer-attached mutex and exception-aware mutex*. In a *timer-attached mutex* a timer is attached to the mutex, thus guaranteeing that the mutex will be released after a predetermined time if a release operation has not been invoked before. An *exception-aware mutex* is a technique that ensures that a mutex gets released when an exception occurs.

4. The Human Factor

> A machine is not easy-to-program if it is not easy-to-teach ... education for parallelism has become a key benchmark. Namely, for parallelism, education is CS research.
>
> Uzi Vishkin, 2009 [61]

A revolution without revolutionaries is like a human without oxygen. The parallel revolution needs scientists, theoretical and practical, who will lead the computer science community to the next era of computing. These scientists have to agree on a bridging model that will guide the computer science practitioners how to build

manycore architectures together with software systems and algorithms. But first and foremost, there is a need to educate the next generation of computer scientists and engineers that will carry the parallel revolution in the years to come. A recent panel discussion of top scientists regarding the future of manycore computing emphasized the need for education:

''Education for thinking in parallel as early as possible in one's career is very important. It was clear to all that regardless of how manycore programming will end up looking, CS must take immediate action toward educating a new generation of people about parallelism. This education scope must soon reach the level of parallel computing education available during the 1990s, but should not stop at that. To the extent possible the education for parallelism should be pushed as early as possible in CS education: to lower division undergraduates and K-12 education.'' [62].

This section is a discussion about the education and training of parallel computing and programming and a brief wish list of the most important issues that must not be overlooked by the parallel computing research community

4.1 Teaching Parallel Programming Today

Many kinds of parallel computing and programming courses are offered by the academia. Most of them are introductory courses, while courses that focus on specific domains such as parallel algorithms or parallel databases are almost nonexistent. Some courses are limited to specific research parallel programming models such as BSP [17, 18] and XMT [63], while others are limited to commercial products such as MPI [64] and OpenMP [7, 8].

The parallel revolution calls for modifying almost any course in the computer science curriculum. The prefix *parallel* can be added to any topic: parallel architectures; parallel OS, parallel algorithms, parallel languages, parallel datastructures, parallel databases, and so on. However, changing the entire computer science curriculum at once is a radical step and is not recommended. On the other hand, lecturers need to prepare their students today for the parallel era. Therefore, it is recommended that they start with a broad introductory parallel computing course of representative topics, theoretical and practical, that teach what parallel computing is all about [65].

We concluded three main wishes from our students expectations. First, the students wish to have a broad view of parallel computing. Second, the students want to understand why parallel programming is difficult. And third, they are eager to try it and wish to have hands-on training. From a survey of the courses offered in universities and colleges around the world we learned three main things. First, most of the courses are based on textbooks that do not teach the students how to think in parallel. Second, the textbooks and the courses avoid discussion of the traps and

pitfalls of parallel programming. And third, the courses cover only a few subjects and usually focus on parallel architectures, parallel programming models, and parallel algorithms.

As for the textbooks, there are many of them, but they all offer very nearly the same topics. The most popular textbooks do not cover topics such as data-dependency, load-balancing, and scheduling. They do not mention important areas such as parallel databases even though all of the commercial databases today are parallel, and technologies such as data-mining and data-warehouse could not exist without large-scale parallel machines. There are no textbooks that can be considered as the ''bibles'' of parallel computing and the parallel programming in the way that Hennessy and Patterson's textbook plays for sequential computer architecture [66], and Cormen et al.'s textbook plays for sequential algorithms [67].

4.2 A Wish List

Writing a scalable and portable application for manycore systems will be impossible task unless the collaboration between the academic research communities and the computer industry can provide appropriate development environments and tools for the mainstream programmer. Language designers, as well as designers of parallel operating systems, compilers, and computer architectures should keep in mind that the principal users of their products are programmers and thus they must design their products from the programmer's point of view. With this in mind, we offer the following wish list:

- *A bridging model*: Parallel programming models should be based on a computational model that is suitable for developing manycore programs. Such a model does not need to be the prefect model but it should present a broad consensus of the community and play a role similar to Von Neumanns model that plays for the sequential computing.
- *Portability*: Software should be able to run unchanged, with high performance, on any general-purpose manycore architecture. However, programming for portability should get higher priority than programming for high performance.
- *Scalability*: The performance of manycore systems, hardware and software, should be able to scale up from a few core to several hundred cores and also be able to scale down from several hundred cores to a few cores.
- *Predictability*: The performance of applications on manycore architectures should be predictable through the use of a simple cost model.
- *Ease of Programming*: Programming of manycore applications should be based on a simple and intuitive parallel programmer model. Such a model

should support high-level abstraction that hides the details of the target manycore processor architecture from the programmer.

- *Standardization*: Standards have several purposes besides the obvious and economically very important one of portability. One such purpose is the setting of quality standards in, for example, minimum levels of generality, consistency, and naturalness in languages. The Multicore Associations Communication API (MCAPI) is one such effort [68]. MCAPI is a communications API for messaging and streaming. It is targeted toward intercore communication in a multicore chip. Accordingly, a principal design goal of MCAPI is to serve as a low-latency interface leveraging efficient on-chip interconnects in a multicore chip.
- *Automatic parallelization*: State-of-the-art compilers can achieve less than 10% performance improvement by using automatic parallelization. Three decades of research in the area of parallelizing compilers have yielded a better understanding of the stumbling blocks that inhibit efficient automatic parallelization. Achieving superlinear speedup by automatic parallelization seems unreasonable goal, but compilers should improve their ability to identify code that can be parallelized and then do the job of parallelizing it without manual intervention from programmers.
- *Debugging tools*: Parallel programming is not only hard and tedious demanding different thinking and creativity, but also a very error-prone one.

The lack of multicore programming tools for mainstream developers is perhaps the biggest challenge the industry faces today. Debugging tools are improving all the time but lack sophistication. They demand from the programmer a broad knowledge about the details of the underlying architecture and familiarity with the hardware counters. Debugging tools should be able to analyze the code and report on bugs and potential bugs while freeing the programmer from the need to be hardware experts.

5. Conclusions

> Hey, the world has changed. The La-Z-Boy approach isn't going to work anymore. You can't just sit there, waiting for your single processor to get a lot faster and your software to get faster, and then you can add the feature sets. That era is over. If you want things to go faster, you're going to have to do parallel computing.
>
> David Patterson, 2007 [43]

Parallel computing is rapidly entering mainstream computing, and multicore processors can now be found in the heart of supercomputers, desktop computers,

and laptops. Consequently, applications will increasingly need to be parallelized to fully exploit the multicore processor throughput gains that are becoming available. However, the promising potential of future manycore processors will be greatly diminished if the industry cannot overcome certain hardware and programming challenges.

This chapter presents the main hardware and software challenges that designers of manycore architectures, alongside designers of software tools, have to face. However, there are more issues, such as load balancing, scheduling, thread-safety, thread-affinity, and performance metrics, that are not covered in this chapter.

Chip makers and system builders have begun efforts to educate developers and provide them with better tools for multicore programming. But many of the tools available are still works in progress. Currently, the responsibility to bridge the gap between hardware and software to write better parallel programs may ultimately lie with developers. Many programmers are not up to speed on the latest developments in hardware design. They should study chip architectures to understand how their code can perform better. This is not a desirable situation. Parallel programming should be as simple and productive as sequential programming.

REFERENCES

[1] J. von Neumann, First Draft of a Report on EDVAC, Moore School of Electrical Engineering, University of Pennsylvania, Pennsylvania, 1945, June 30.
[2] J. Van Der Spiegel, J.F. Tau, T.F. Ala'illma, L.P. Ang, The ENIAC: History, operation and reconstruction in VLSI, in: R. Rojas, U. Hashagen (Eds.), The First Computers, History and Architectures, MIT press, Cambridge, MA, 2002, pp. 121–178.
[3] W. Aspray, The institute for advances study computer: A case study in the application of concepts from the history of technology, in: R. Rojas, U. Hashagen (Eds.), The First Computers, History and Architectures, MIT press, Cambridge, MA, 2002, pp. 179–193.
[4] M. Marcus, A. Akera, Exploring the architecture of early machine: The historical relevance of the ENIAC machine architecture, IEEE Ann. Hist. Comput. 18 (1996) 17–24.
[5] J.P. Eckert, J.W. Mauchly, H.H. Goldstine, J.G. Brainerd, Description of the ENIAC and Comments on Electronic Digital Computing Machines, Moore School of Electrical Engineering, University of Pennsylvania, Pennsylvania, 1945, November 30.
[6] A. Marowka, Parallel computing on any desktop, Commun. ACM 50 (9) (2007) 74–78.
[7] OpenMP Architecture Review Board, OpenMP Application Program Interface, 2008. Version 3.0, May.
[8] B. Chapman, G. Jost, R. van der Pas, Using OpenMP Portable Shared Memory Parallel Programming, MIT Press, Cambridge, MA, 2007.
[9] T. Mattson, J. DeSouza, How Fast is Fast? Measuring and Understanding Parallel Performance, 2008. Intel Webinar.
[10] K. Olukotun, L. Hammond, The future of microprocessors, ACM Queue (2005) 27–34 September.
[11] H. Sutter, The free lunch is over: A fundamental turn toward concurrency in software, Dr. Dobb's J. 30 (3) (2005) 202–210.

[12] H. Sutter, J. Larus, Software and the concurrency revolution, ACM Queue 3 (7) (2005) 54–62.
[13] A. Marowka, Think parallel: Teaching parallel programming today, IEEE Distrib. Syst. Online 9 (8) 2008.
[14] S. Hambrush, Models for Parallel Computatio, in: Proceeding of ICPP Workshop on Challenges for Parallel Processing, pp. 92–95.
[15] D. Skillcorn, D. Talia, Models and languages for parallel computation, ACM Comput. Surv. 30 (2) (1998) 123–169.
[16] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, LogP: Towards a realistic model of parallel computation, in: Proceeding of 4th ACMSIGPLAN Symposium on Principles and Practices of Parallel Programming, pp. 1–12.
[17] L.G. Valiant, A bridging model for parallel computation, Commun. of ACM 33 (8) (1990) 103–111.
[18] H.R. Bissling, Parallel Scientific Computation: A Structured Approach Using BSP and MPI, Oxford Press, Oxford, 2004.
[19] T.S. Kuhn, The Structure of Scientific Revolutions, first Ed., University of Chicago Press, Chicago, 1962, p. 168.
[20] Parallel@Illinois. University of Illinois at Urbana-Champaign, http://www.parallel.illinois.edu/.
[21] Pervasive Parallelism Laboratory at Stanford University, http://ppl.stanford.edu/wiki/index.php/Pervasive_Parallelism_Laboratory.
[22] Parallel Computing Laboratory at Berkeley, http://parlab.eecs.berkeley.edu/about.html.
[23] K. Asanovic, et al., The Landscape of Parallel Computing Research: A View from Berkeley, University of California, Berkeley, CA, 2006. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html, Technical Report No. UCB/EECS-2006-183, December 18.
[24] http://www.highproductivity.org/.
[25] A. Marowka, Portability of Parallel and Distributed Application, The Hebrew University of Jerusalem, Israel, 2000, PhD Dissertation.
[26] A. Marowka, I. Yarchi, M. Bercovier, Scalable portability evaluation of high performance applications, Int. J. Comput. Appl. 7 (1) (2000) 39–47.
[27] A. Marowka, M. Bercovier, A scalable portability model for parallel computing, J. Parallel Distrib. Comput. Pract. (PDCP) 3 (3) (2000) 133–156.
[28] A. Marowka, Portability of Parallel and Distributed Applications: Is it Possible to Build a Portable and Scalable Parallel Application? VDM Verlag, Germany, 2009.
[29] E.A. Brewer, High level optimization via automated statistical modeling, ACM SIGPLAN Notices 30 (8) 1995. Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'95).
[30] E.A. Brewer, Portable High-Performance Supercomputing: High-Level Platform-Dependent Optimization, 1994. MIT Ph.D. Dissertation, September.
[31] T. Hey, J. Ferrante, Portability and Performance of Parallel Processing, Wiley, Chichester, NY, 1994.
[32] J. Demmel, J. Dongarra, et al., Self-adapting linear algebra algorithms and software, Proc. IEEE 93 (2) (2005) 293–312.
[33] Skeletal Parallelism Homepage, www.dcs.ed.ac.uk/home/mic/skeletons.html.
[34] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Pozo, C. Romine, H. Van, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, PA, 1993.www.netlib.org/utk/papers/etemplates/paper.html.
[35] D.G. Feitelson, L. Rudolph, Toward convergence in job schedulers in parallel supercomputers, in: D.G. Feitelson, L. Rudolph (Eds.), Proceeding of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing, April. *LNCS* 1162 (1996) 1–26. Springer, Berlin.

[36] R. Merritt, HP calls for on-chip optics, EE Times (2009) January.
[37] M. LaPedus, Intel lists five challenges for IC scaling, EE Times (2009) February.
[38] I. Young, E. Mohammed, J. Liao, A. Kern, S. Palermo, B. Block, M. Reshotko, P. Chang, Optical I/O Technology for Tera-Scale Computing, in: International Solid State Circuits Conference (ISSCC).
[39] A. Adl-Tabatabai, C. Kozyrakis, B. Saha, Unlocking concurrency, ACM Queue (2007) 24–33, January.
[40] M. Herlihy, E. Moss, Transactional memory: Architectural support for lock-free data structures, in: Proceedings of the 20th Annual International Symposium on Computer Architecture, San Diego, CA, May.
[41] B. Cantrill, J. Bonwick, Real-world concurrency, ACM Queue (2008) 16–25, September.
[42] A. Marowka, Empirical Analysis of Parallelism Overheads of CMPs, in: Proceeding of Eighth International Conference on Parallel Processing and Applied Mathematics (PPAM, September 2009), LNCS, Springer.
[43] K. Olukotun, A Conversation with Hennessy and Patterson, ACM Queue (2007) 14–22, January.
[44] W.J. Dally, B. Patrick, Principles and Practices of Interconnection Networks, Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann, San Francisco, CA, 2004.
[45] V. Kumar, A. Grama, A. Gupta, G. Karipis, Introduction to Parallel Computing, Benjamin/Cummings, Redwood City, CA, 1994.
[46] D. Culler, J.P. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann, San Francisco, CA, 1998.
[47] S. Akhter, J. Roberts, Multi-Core Programming: Increasing Performance Through Software Multi-threading, Intel Press, 2006.
[48] A. Mendelson, J. Mandelblat, S. Gochman, A. Shemer, R. Chabukswar, E. Niemeyer, A. Kumar, CMP implementation in systems based on the Intel Core Duo Processor, Intel Technology Journal 10 (2) 2006.
[49] A. Marowka, Performance of OpenMP Benchmarks on Multicore Processors, in: 8th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), LNCS Proceeding, vol. 5022, pp. 208–219 Agia Napa, Cyprus, June 9–11.
[50] D.E. Knuth, A. Binstock, Interview with Donald Knuth, InformIT, 2008, April 25.
[51] D. Miles, B. Leback, D. Norton, Optimizing Application Performance on Cray Systems with PGI Compilers and Tools, in: Proceedings of CUG.
[52] Intel Thread Checker, http://software.intel.com/en-us/intel-thread-checker/.
[53] Intel Composer, http://software.intel.com/en-us/intel-parallel-composer/.
[54] R. Allen, K. Kennedy, Optimizing Compilers for Modern Architectures: A Dependence-Based Approach, Morgan Kaufmann, San Francisco, CA, 2001.
[55] A.J. Bernstein, Analysis of programs for parallel processing, IEEE Trans. Electron. Comput. 15 (5) (1966) 757–763.
[56] L. Lamport, How to make a multiprocessor computer that correctly executes multiprocess programs, IEEE Trans. Comput. C-28 (9) (1979) 690–691.
[57] S.V. Adve, K. Gharachorloo, Shared memory consistency models: A tutorial, IEEE Comput. 29 (12) (1996) 66–76.
[58] E.W. Dijkstra, Cooperating Sequential Processes, Academic Press, London, 1968. Programming Languages.
[59] M. Bull, D. O'Neill, Microbenchmark Suite for OpenMP 2.0, in: Proceedings of the Third European Workshop on OpenMP (EWOMP01), Barcelona, Spain, pp. 41–48.

[60] E.A. Lee, The Problem with Threads, Electrical Engineering and Computer Sciences, University of California, Berkeley, 2006, Technical Report No. UCB/EECS-2006-1.
[61] U. Vishkin, The PRAM-On-Chip Proof-of-Concept, 2009. Workshop on Theory and Many-Cores (T&MC).
[62] Workshop on Theory and Many-Cores (T&MC), http://www.umiacs.umd.edu/conferences/tmc2009/, University of Maryland, College Park, Maryland, 2009, May 29.
[63] U. Vishkin, S. Dascal, E. Berkovich, J. Nuzman, Explicit multi-threading (XMT) bridging models for instruction parallelism (extended abstract), in: Proc. 10th ACM Symposium on Parallel Algorithms and Architectures (SPAA), http://www.umiacs.umd.edu/users/vishkin/XMT/index.shtml.
[64] M. Snir, et al., MPI: The Complete Reference, MIT Press, Cambridge, MA, 1996. http://www.mcs.anl.gov/research/projects/mpi/.
[65] A. Marowka, Think parallel: Teaching parallel programming today, IEEE Distrib. Syst. Online 9 (8) 2008.
[66] J.L. Hennessy, D.A. Patterson, Computer Architecture, Fourth Ed: A Quantitative Approach, Morgan Kaufmann, San Francisco, CA, 2006.
[67] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, McGraw-Hill, 2003.
[68] J. Holt, A. Agarwal, S. Brehmer, M. Domeika, P. Griffin, F. Schirrmeister, Software standards for the multicore era, IEEE Micro (2009) 40–50 March/April.

Illusion of Wireless Security

ALFRED W. LOO

Lingnan University, Hong Kong, China

Abstract

This chapter examines the misunderstanding of wireless security from user behavior perspectives. Many users believe their wireless connections are safe and they do not realize the serious consequences of possible security breaches. These problems must be rectified quickly due to expanding use of wireless facilities.

Corporations also underestimate the potential dangers. IT managers do not provide enough education and support for users' wireless connections at home or public places. IT managers are not paying enough attention to issues and that pose a serious threat. Urgent action is required in light of the recent high-profile security breaches due to ignorance and negligent use.

Most threats come from the ignorance of users, the inactive attitudes of corporations, and the improper implementation of security features by wireless devices manufacturers. Thus manufacturers need to develop more secure products by devoting more time to the development of security features and design of human–computer interface (HCI). As technology alone will never be able to solve all security problems, enhancement of the coordination between employers and end-users is required.

In addition to discussing the truths and myths, countermeasures are presented to rectify these problems. This chapter differs from other security papers in that the minutiae of technical terms (such as algorithms, details of data frames, etc.) are avoided as far as possible so it can reach a wider audience.

1. Introduction 120
   1.1. Awareness Level of Users 121
   1.2. Organization of This Chapter 122
2. Wireless Routers 122
   2.1. Functions of Wireless Routers 123

ADVANCES IN COMPUTERS, VOL. 79
ISSN: 0065-2458/DOI: 10.1016/S0065-2458(10)79003-3

2.2. The Myths 126
2.3. The Truths 128
2.4. The Consequences 132
2.5. Countermeasures 134
3. Smart Phones 137
3.1. Overview of Bluetooth Technology 137
3.2. Illusions 140
3.3. Reasons for the Misunderstandings 141
3.4. Possible Attacks 141
3.5. Consequences 142
3.6. Phone Virus Development 143
3.7. Countermeasures 145
4. Threats of Public Wireless Networks 147
4.1. War Driving 149
4.2. Possible Attacks 150
5. Illusions of Encryptions 153
5.1. Basic Concept of Encryption 154
5.2. Other Applications of Encryption 157
5.3. Illusions 158
6. Conclusion 162
Acknowledgments 163
References 166

1. Introduction

The Her Majesty's Revenue and Customs (HMRC) department of the British Government has lost the personal details of 25 million people in a security breach [1]. It caused huge embarrassment and the UK Prime Minister, Mr. Gordon Brown, who has been forced to apologize. The chairman of HMRC, Mr. Paul Gray, resigned. Up to 7 million families have been warned to be on alert for attempts of fraudulence.

Certainly, it was a shock to members of these families. Many computer security professionals were also very surprised as this breach did not involve any high-tech hacking. It was simply a user mistake.

Two computer disks were sent from Her Majesty's Revenue and Customer department to National Audit Office in London via internal mail. However, these

disks never arrived at their destination. Data on the disks include names, addresses, date of birth, National Insurance numbers, and bank account details. In addition to putting bank accounts under threat, the missing information could be used to impersonate the victims to buy goods or take out a loan. These pieces of information are more valuable than credit card numbers in the black market and can be sold for high prices.

We expect this kind of organization to be aware of security threats and their serious consequences as they are processing very sensitive and important data. An organization of such a size should be able to employ a team of security experts. Researchers in IT security are very active and papers in this area are abundant [2–4]. We tend to believe security experts in government have adopted all possible measures and such breaches are highly unlikely. However, this incident has once again proved the vulnerabilities of highly secured computer systems with ignorant users.

Organizations with fewer resources are even more vulnerable than the above-mentioned department. The incident is a wakeup call for every organization to review security measures and policies for their computer users.

1.1 Awareness Level of Users

The strength of the security of a computer system is always measured by its weakest component. The weakest components in most systems are the end-users, particularly when they are accessing the corporation's databases with wireless facilities at home or in public areas. The reasons are simple:

- Wireless facilities are relatively new for end-users and there is a great deal of misunderstanding over their security features.
- While computer facilities within the organization are protected by computer experts, the connections from employees' homes and public places are not.
- Research papers and books on wireless security are written for technical people. Most users, including some computer professionals, are not able to understand these materials.
- Wireless products were not widely available in the consumer markets 10 years ago. Wireless technology courses were rare (except for Electronics/Electrical Engineering majors) in universities at that time and thus even computer professionals who graduated in the past decade have very little knowledge in the modern wireless technologies.

Many nontechnical users are using the following wireless facilities today:

- Wireless routers
- Smart phones
- Public Wi-Fi networks

Improper use of the above facilities poses serious threats to both individuals and corporations. Recent experiments [5–9] indicated the low level of public awareness of potential dangers. Furthermore, both individuals and corporations underestimate these dangers and they do not realize the serious consequences. Security threats are simply ignored. This problem must be rectified quickly due to widespread use of these wireless devices.

1.2 Organization of This Chapter

Most threats come from ignorance of users, inactive attitudes of corporations, and improper implementation of security feature from wireless devices vendors. Examples of possible hacking and damages are presented in the rest of this chapter. These issues are discussed in a nontechnical way so it can reach a wide range of audience. Thus, this chapter differs from other security papers in that technical terms (such as algorithms, details of data frames, etc.) are minimized as far as possible.

The remainder of this chapter is organized as follows. Section 2 presents the threats of wireless routers and the possible consequence. Wireless communication standards and terms are introduced to readers as background information. Countermeasures which can minimize the threats are discussed. Section 3 presents the security issues associated with ''smart'' phones as many people are not aware of these problems. An overview of Bluetooth technology is presented. Other wireless threats are discussed in further detail and solutions are provided. Section 4 presents the threats of using public wireless fidelity (Wi-Fi) networks. Section 5 discusses the illusions of encryption. An overview of encryption is provided. We summarize our discussion of this chapter in Section 6.

2. Wireless Routers

Wireless routers began to be available about a decade ago. The cost has been decreasing quickly in the past few years due to mass production. Now they are inexpensive—most cost less than US $60. It is extremely easy to install a wireless router. The wireless router provides the following conveniences to the users:

- It is quite common that families have several computers. These computers are usually in different rooms. To share broadband communication, the installation

of network cables would be a time-consuming and expensive task, depending on the distances between rooms. Wireless routers solve this problem easily. Theoretically, a typical wireless router can support 256 computers simultaneously.

- Users can move their computers easily around the home, as long as they remain within the coverage of the wireless signal. They do not need to rearrange network cables. For example, you have installed your notebook computer in the bedroom. On a particular day, you are working on an urgent project for the company. You continue to work with your computer at mid-night but you do not want to disturb your spouse. You can simply take your notebook computer to the living room. You will still be able to access the Internet via the wireless facility and save hours of cable reconnection.

2.1 Functions of Wireless Routers

When a computer tries to access the Internet, it needs to connect to the computer of an Internet service provider (ISP) first. Usually user needs to log in with a valid account and password which is provided by the ISP. After the login process, a unique Internet protocol (IP) address will be assigned to the user's computer as in Fig. 1. The user will type the uniform resource locator (URL) of the remote Web server on his/her Internet browser. This URL will then be sent to the ISP. The ISP's computer is the one that actually contacts the remote Web server according to the URL. The remote Web server will send the Web page to the ISP first. The ISP will then convey the Web page to the user as in Fig. 2, the traditional wired communication.

In a wireless connection, the ISP computer works in the same way as in the wired communication. However, a wireless router is added in between the user's computer and the ISP's computer as in Fig. 3. We can consider the router as a small dedicated computer. Its function is to convey messages between the user's computer and the ISP's computer. The ISP's computer does not know whether it is talking to a computer or a router. It does not know how many computers are using the router as well. Thus the wireless router is acting as a coordinator between the ISP and the user.

2.1.1 Overview of IEEE802 Standards

Wireless routers use the IEEE802.11 standards to communicate with user's computers. These standards were developed by the IEEE (Institute of Electronic and Electrical Engineers) which is a nonprofit making organization. IEEE (often pronounced as eye-triple-E) has developed many different communications standards to make sure devices from different vendors can communicate with each other.

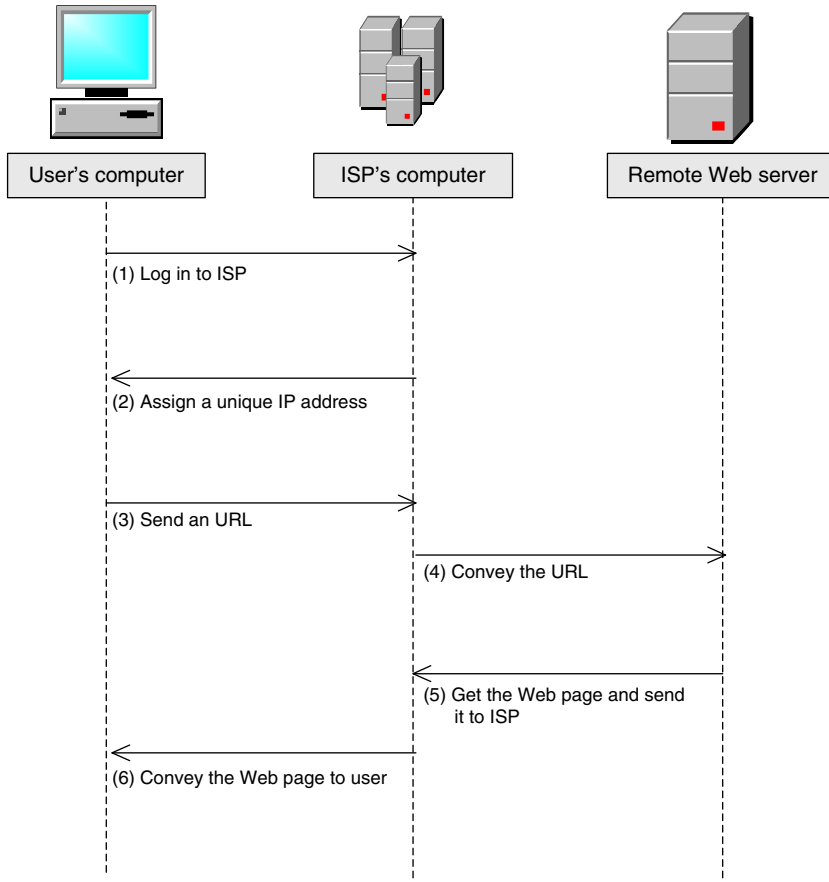

FIG. 1. Login process.

The first design began in 1990 and it took 7 years to be completed. It uses the 2.4 GHz radio band for communication. Users do not need a license to use this band in most countries. This is one of the reasons that this standard has become so popular. The standard specifies the communication between wireless client and the access points. It also defines the optional use of encryption. However, there are some well-known security weaknesses in this standard and the maximum speed is only 2 Mbps. This version is obsolete now.

To overcome the shortcomings of the first design, IEEE workgroups have developed new IEEE802.11 standards. They differentiate these newer standards by

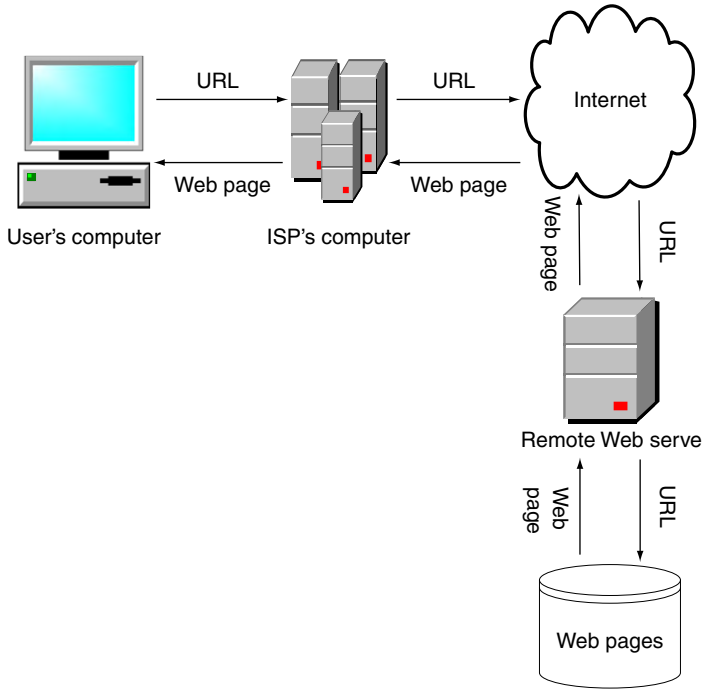

FIG. 2. Web surfing with wired communication.

appending a letter behind the 802.11. Although there are large numbers of new standards, only four standards have products in the market now. They are IEEE802.11a, IEEE802.11b, IEEE802.11g, and IEEE802.11n.

Before 2003, the most popular standard was the IEEE802.11b. Under this standard, the data transfer rate is improved to 11 Mbps and the range to 100 m while still using the 2.4 GHz radio band. Access points and network cards using this standard are inexpensive. Devices from different vendors are extremely compatible with each other. Installation is easy and there is widespread use in small offices and homes. As the standard is slower than 11a and 11g, 11b products have been phasing out of the market in recent years. However, there are still a large number of 11b devices in use today.

On the other hand, IEEE802.11a offers much higher speed (up to 54 Mbps). It uses the 5 GHz radio band so it avoids interference from electronic devices (such as microwave ovens, cordless phones, etc.). A wireless signal spectrum is presented in Fig. 4. However, this radio band is not available for unlicensed

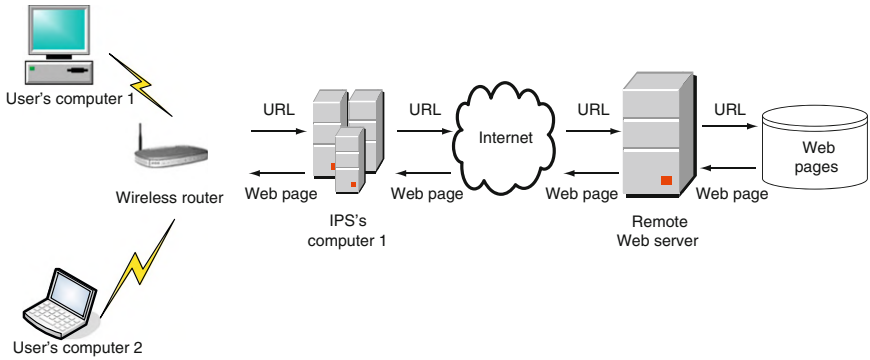

FIG. 3. Web surfing with wireless communication.

use in all countries. Another problem is that IEEE802.11a devices are not backward compatible to the slower IEEE802.11b. This problem may be a barrier deterring organizations which already have IEEE802.11b devices from deploying 802.11a.

IEEE802.11g provides a higher speed (up to 54 Mbps) and it is backward compatible to 802.11b. Organizations do not need to replace all existing 802.11b devices when they add new 802.11g technologies. The first products under this standard became available in January 2003. It also uses the 2.4 GHz radio band.

The next generation of IEEE802 is 11n which still uses the 2.4 GHz radio band. It provides higher speed (up to 300 Mbps) and better security. At the time of writing this chapter, IEEE802.11n is still in a draft form. However, many vendors are selling ''11n draft'' routers now in order to gain a better marketing position in the future. Many people believe that the difference between the final version and the draft will be little. Most vendors also hope that their ''draft'' products will be compatible to the final version by simply updating the firmware on the devices in the future.

The differences between these four standards are summarized in Table B.I.

2.2 The Myths

The author had a recent discussion with one of his friends. The friend is a seasoned application programmer and has a master's degree. He is using a wireless router at home and believes his wireless connection to be secure. It is not. If an application programmer puts his faith in myths, we cannot expect anyone with less computer knowledge to be aware of the security problems.

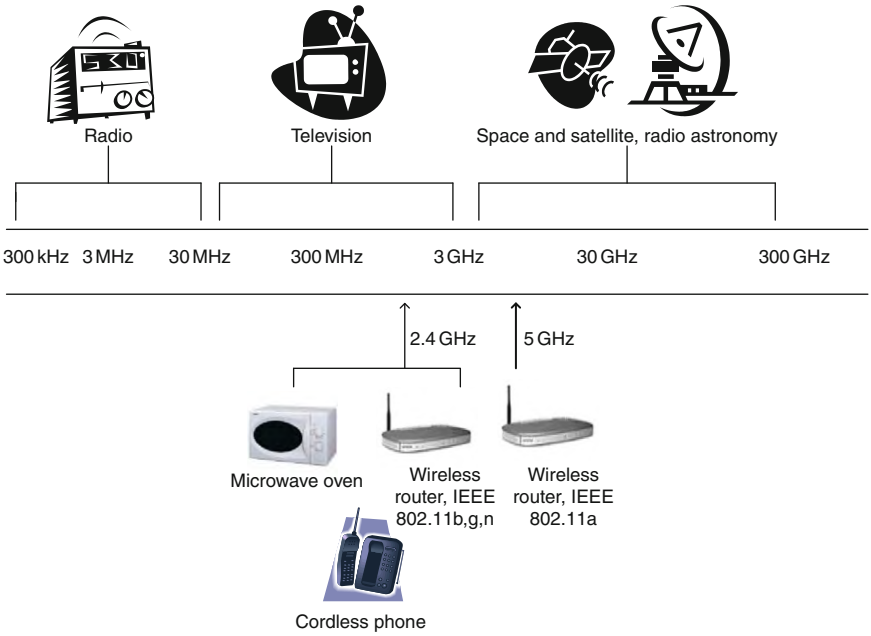

FIG. 4. Wireless signal spectrum.

In theory, nothing can be 100% safe. For example, if you want to protect your house from burglary, you can install a good lock on your door. A determined thief can still break into your house provided he has the right equipment and enough time. Even if he cannot break the lock, he can break the door. Alternatively he can break the window, drill a hole on the wall, dig a tunnel, use explosives, etc. However, all these methods will force him to take a longer time to finish the task. He will make more noise. Carrying more equipment will increase the chance that he will be detected or even arrested before or after the task. A rational thief will compare his cost with the estimated valuables in your house. He might decide that it is not worthwhile to break into your house provided you have the right degree of security.

Wireless communication is very similar to the above scenario. However, while many users believe their doors, at least, are locked, the truth is the doors are not even closed and are actually wide open [2,10,11]. Intruders can simply walk in. A wireless router user might believe the following:

- Wireless routers have been in the consumer market for several years. A lot of people are using them so it must be a mature and safe technology. There are enough security features in the router so their communications are automatically protected.
- It is time consuming to hack a computer so attackers will go after the computers of corporations. There is nothing valuable enough in home communications to attract hackers. Users have little to lose. There will not be any serious consequences even if their transmissions are intercepted by hackers.

2.3 The Truths

Most wireless routers have some security features. However, these features are optional and usually turned off. As many users do not have the security knowledge, they do not know how to turn these features on.

2.3.1 Stealing Bandwidth

The installation of a router is quite easy and users can complete it within minutes. A small installation program will ask the user to type in the account name and password of their ISP during the initialization process. Users might believe that they are protected by the password afterwards; indeed, the account name and password are stored in the router. When users turn on their computers and routers, the router will log in to the ISP automatically as in Fig. 5. The computers will then communicate with the router directly and access the Internet through the router. The router will not ask the computer to supply an account name and password again. This means the hacker's computer can connect to the router without any knowledge of the account.

The hacker does not need any hacking tools as nowadays' operating systems (e.g., Microsoft Windows XP) can find any nearby routers and connect to them almost automatically. Hackers can connect to the Internet and perform illegal operations such as hacking other computers, spreading viruses, organizing terrorist activities, etc. He/she does not need to get into the users' house or plug in any cable. He might be sitting inside a car [12–14] which is parked on the road near the user's house. Hackers might also be the neighbors of users. All he needs is a notebook with a wireless LAN card. It is extremely difficult for the user to detect this hacking process. As the hacker does not need to supply any account name, password, IP address, or any identification to the ISP, it is impossible to trace the hacker afterwards.

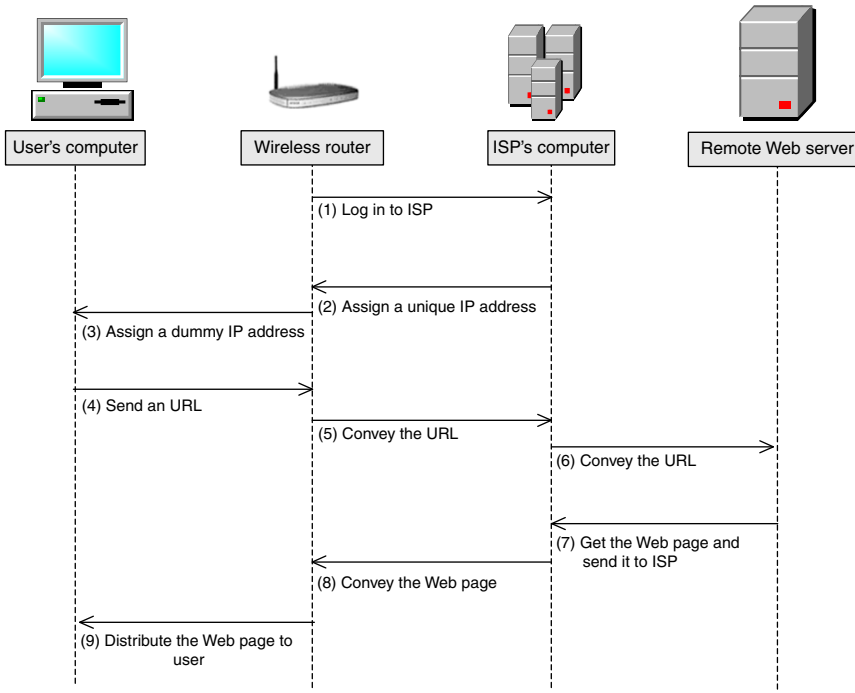

FIG. 5. Operation of a wireless router.

Hacking can take place automatically if proper software is installed on the computer. For example, a hacker could leave their computer on in a van while masquerading as technicians from telephone or electricity companies. They do not need to stay in the van during the hacking processes. Antennas and amplifiers can extend the area of wireless coverage. A study [15] shows that it is possible to attack a target from 20 miles away. This kind of hacking is more difficult to detect (Fig. 6).

As the router is actually sending/receiving messages for the computers, it is impossible for the ISP to detect the number of users. This creates headaches for the users and Governments. When the police come to the doors of the legitimate users, they will have a hard time explaining what is going on. Governments will also have problems in identifying the hackers from the ISP's records. Even if the hacker

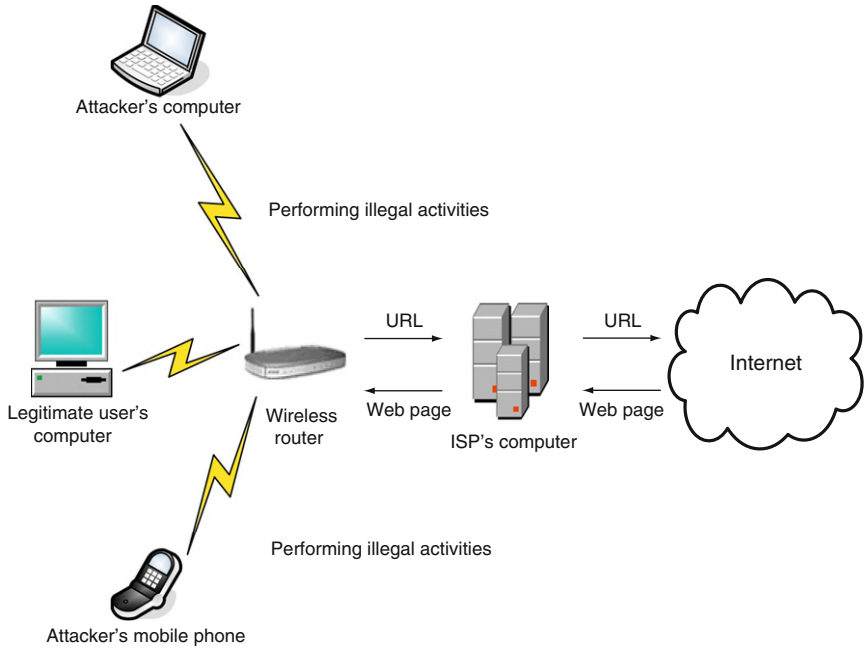

FIG. 6. Stealing bandwidth.

is caught on the scene, it will be difficult to collect evidence, prosecute, and convict him/her in the court as there need be no particular hacking tools (hardware/software) on his/her computer.

There is another illusion in the initiation process. Many routers have a default account ''admin'' (with the same default password ''admin''). Users can change the password or create a new account. They might believe this is another level of protection. Actually this account is used only when the users want to view or revise the settings in the router. Hackers do not need this piece of information to access the Internet through the router. This illusion only gives users a false feeling of security.

2.3.2 Eavesdropping

The most well-known problem of wireless communication is that it is susceptible to anonymous attackers. The attacker can use very simple equipment to intercept the wireless signals and decode the information being transmitted to the access point.

Such equipment can be just the devices (e.g., notebook computer and smart phones as in Fig. 7) which are used to access the network. These devices can be configured to intercept all traffic on a network so long as the attacker is in the proximity of the transmitter. Wireless packet analyzers [16,17] for such purposes can easily be obtained from the Internet.

The use of antennas and amplifiers enables longer distance attacks. Wireless signals may travel outside a building. Recent studies show that it is feasible to attack a target from 20 miles away [15]. It is almost impossible to detect and prevent such attacks.

As a notebook computer or smart phone is so small and there is no physical wired connection, an attacker can hide anywhere within the coverage area. An attacker can monitor transmissions from parking lots and nearby areas. Even if the user knows an attacker is around and he/she is hacking the network, it is still almost impossible to identify the attacker.

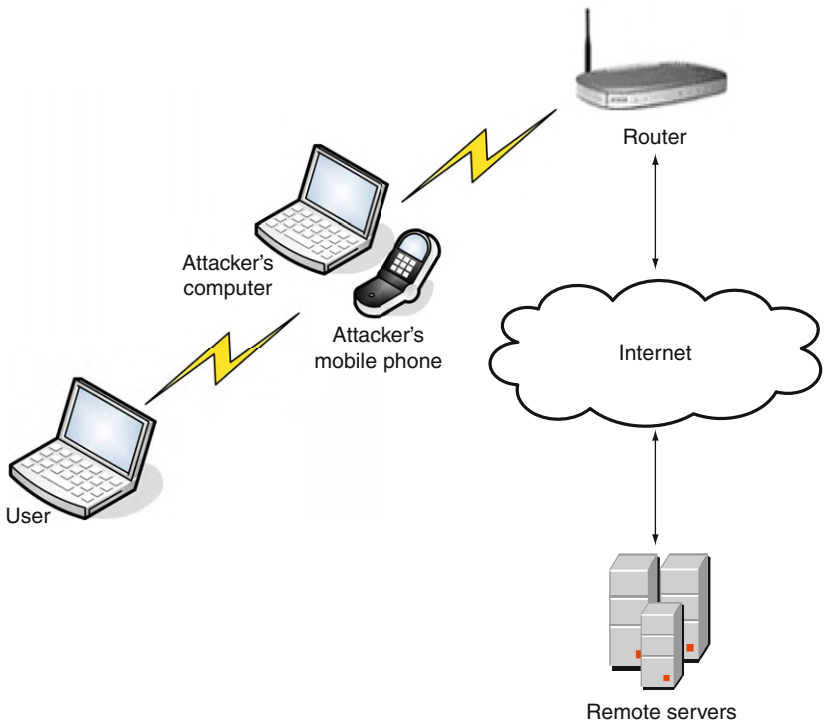

FIG. 7. Eavesdropping.

2.4 The Consequences

Many companies have developed applications with Internet technologies. These applications offer many advantages. Their employees can access the databases of the company with an Internet browser such as Internet Explorer or Firefox. There is no need to install special programs on the end-users' computers. In the past, updating application programs on the end-users' computers was tedious and error prone. These tasks have been minimized or eliminated as all programs are held on the servers. Naturally, employees can also access the systems with their home computers so they can continue to work at home.

Such advantages are accompanied by additional risks. Many systems rely on only account name and password to authenticate users. If the hacker can get this information while the user is accessing corporate databases, every security feature installed by the corporation will fail and the unauthorized intrusion will go undetected.

Some corporations protect their login pages with secure socket layer (SSL) or other similar techniques. Such traditional methods for wired communication can also be effective for wireless networks. For example, the Web site sends a public key to the user for encryption for every session. The users are pretty safe if the corporation is using such techniques. It is useless for hackers even they can intercept the transmissions. Users might believe that further encryption is not necessary. However, users will also access Web sites which do not require high levels of security. A patient hacker can obtain hints from these nonsecured communications of the users.

It is common for users to have multiple accounts. For example, after purchasing new products (such as mobile phones, TV, etc.) many companies allow their customers to send the warranty information through their Web sites. Customers need to create an account in order to obtain updated information about their products. Many Web sites which provide free information will also ask their readers to create new accounts. As these Web pages do not involve any money or confidential information, encryption techniques might not be used. Attackers can easily obtain the account names and passwords of a user if this information is sent over a wireless connection without proper protection. The attacker can then masquerade as a legitimate user to attack, for example, the user's corporation.

In theory, users should use different passwords (and account names) for different accounts [18]. However, it is similar to asking our younger generation not to have sex before marriage. In practice, many people use the same password for multiple purposes as it is extremely difficult to remember all accounts and passwords, especially when some accounts are not used frequently. If the account name and

password are intercepted in an unsecured transmission, the hacker can spoof the user and break into the company's database.

Even if the users use different passwords, it is still dangerous not to protect passwords for nonsecured Web sites. By conducting analysis on users' communications, hackers can find out the pattern of users in composing the passwords. It is difficult for humans to remember a combination of characters and numbers which have no meaning. A common practice is to take words and numbers which they are familiar with and combine them. Let us look at the following example.

One user might use the first four characters of his son's name (e.g., ''davi'' from ''David''), combined with year of his son's birthday (e.g., 1992). To make it a little more difficult to crack, the user reverses the order and add a special character in the middle so the final password is 2991@ivad. This password is not bad for the following reasons:

- It is easy to remember so user does not need to write it down.
- It does not use a complete word from the dictionary so hackers are forced to spend more time in a ''brute-force'' attack (trying many different combinations by varying one character at a time).
- It uses alphabetic, numeric, and special characters so this will maximize the number of possible combinations.
- The password is long enough (i.e., nine characters in this case).

However, an assiduous hacker will find out the pattern if he can intercept one password in nonsecure communications. The attacker can then try the user's corporate account with his daughter's name and year of birthday (by adding or subtracting a small number from the 1992 in the above example). If this does not succeed, the attacker can try his wife's name or something similar. In other words, such information could help to minimize the number of trials needed to break into a computer system.

By eavesdropping to user's communications, hackers can find out the user's favorite or frequently used words and Web sites. This kind of information can also speed up the hacking process. Thus it is important to encrypt all communications even if there is no confidential information in the Web pages.

Finding executives of a target corporation is also easy. The names of executives can be found in their companies' Web sites. Sometimes there are even photos and brief biographies on the same Web page. Hackers can follow the executives' cars and find their residences easily. Armed with this knowledge, the hacker is ready to carry out his job.

2.5 Countermeasures

Users will not be able to solve these problems by themselves due to limited resources and knowledge. Three parties need to work together in order to tackle these threats and minimize any possible damages. These parties are:

- Corporations
- Manufacturers of wireless devices
- Users

2.5.1 The Corporation's Responsibilities

Parents will not leave their babies in the jungle and expect them to protect themselves. Nevertheless, many organizations do not take care of the home Internet connections of their employees. The widespread use of wireless routers of their employees is a threat to many computer systems. There are many ways to attack a wireless communication so it is almost impossible to make it absolutely safe.

There are also basic weaknesses in the current technologies (e.g., WEP [3,19]) which are still being used in many routers. However, we should make the hacker's life as difficult as possible. In other words, we must force the hacker to invest more time, install more hardware or/and software for each attack. This will increase the hacker's risk of being caught. Corporations should urgently review their security measures. Some new measures are easy to implement and we provide a few examples as follows:

- Corporations should educate their employees so they know the risks and countermeasures. From time to time, they can offer short seminars (preferably less than 1 hour) and materials should be explained in layman's terms.
- Corporations should remind users to be vigilant and report suspected cases to authorities as soon as possible. For example, a car is parked around the user's house for a long time, over many days: if the driver is not his neighbor or a legitimate visitor, users should try to find out more information.
- The users should be taught to turn on some easy and common security features of the router. These features can be deployed in a few minutes provided the router manufacturer builds software with good HCIs. It is not possible to discuss all features in this chapter, but we will provide a few examples:
  - Every LAN card has a MAC address. In layman's term, it is a global unique serial number for the LAN card. Users can store the MAC address and ask the router only to accept computers with the right MAC addresses. Using MAC

addresses to identify users is not completely foolproof. If the hacker knows the MAC address, he can install a special program and spoof the address to the router.
  - Communication between computers and routers are transmitted in plain text. Messages can be intercepted easily. The hacker needs only an ordinary computer and wireless LAN card to intercept the signal. The countermeasure is to turn on the security features in the router. Users can specify an encryption key on the router and the computers. Messages between computers and routers will then be encrypted. The keys should be changed frequently as there are weaknesses in encryption technology.
  - Wireless routers broadcast their service set identifier (SSID) to surrounding computers. We can consider the SSID to be the ''name'' of a router. Usually it is a default name assigned by the manufacturer. Computers need this name for proper connection to the router. The user should change this name often and disable the broadcasting of the router. This measure will make the hacking more time consuming.
  - Every router comes with an administrator account that has a default password. Some users do not bother to change this password. Once hackers log into routers, they can modify the settings and/or turn off all security mechanisms. Users should change the password (and account name) to protect the settings.
- Most routers installations are easy. However, some routers come with poor HCIs in the parts which activate the security features. It is difficult for users to select a good router on their own. Corporations should evaluate routers and recommend only products with good security features and human–computer interfaces. If enough corporations are offering these kinds of recommendations, it will force the manufactures to pay more attention to developing these areas.
- Adoption of digital certificates and virtual private network (VPN) should be considered as it will provide higher levels of security in the long run.

2.5.2 The Manufacturer's Responsibilities

There are flaws [3,19,20] in the current security technologies of wireless routers. Security experts are proposing new standards to overcome these problems. However, manufacturers of routers can still improve security while they are waiting for the new standards. Security will never be perfect as hackers can always find new methods to crack systems anyway. In the mean time, some things can be done quickly and easily. Examples of improvements include:

- Router manufacturers should make their products safer by turning on security features as default settings. User should be warned of the risks if they turn off these features.
- Manufacturers should incorporate small resident programs which allow the user to check the numbers of computers connected to the router at any particular time. These programs could be executed in the user's computer or the router (if the router has enough memory).
- Log files should be maintained so users can check whether there is any intruder.
- Wired equivalent privacy (WEP) and Wi-Fi protected access (WPA) are two common methods to encrypt traffic between the device (e.g., laptop computer) and the wireless routers. However, most home routers have the instructions to ignore setting the WEP key when installing, thus adding a level of insecurity. Users should set their own keys to add some level of security.

There are known weaknesses in the WEP encryption. WPA should be used [19,21] if it is available. However, many networks need to support old devices which cannot use new encryption techniques. It is still worth deploying this method.

The current WEP key encryption can be cracked when the hacker collects enough of a user's transmissions. The encryption key should be changed frequently. However, changing keys frequently is easier said than done. As the key resides in both computer and router, it is tedious and a little bit difficult for nontechnical users to do this task. It is more time consuming if the user has more than one computer:

- Manufacturers should provide a program which could change the key on both devices. To make things even easier, the program should be automatically executed when the user turns on the computer. Instead of asking the user for the new key, the program can generate random keys and update all devices periodically. The SSID should be changed in the same way.
- Once a new hacking method is discovered, it (with countermeasures, if any) should be announced in the manufacturers' Web site or sent to the users through e-mail. Updated software and/or firmware (if any) which can combat the new attack should be available for downloading from the manufacturers' Web sites.

2.5.3 User Responsibilities

- Users must follow the security procedures of their employers. It is their duty to safeguard the system by using computers properly.
- Users should invest their time in learning how to use wireless communications safely, such as by attending seminars offered by their employers.

- Users should be vigilant at all times. For example, many computer systems display the last time that users logged in. These records should be checked to detect intruders.

3. Smart Phones

A security experiment known as BlueBag [22, 23] was conducted in March 2006 to test the vulnerabilities of mobile devices. A computer with a Bluetooth sniffing program was hidden in a suitcase that was rolled around train stations, shopping centers, airports, and other public places. The objective was to find out the number of Bluetooth-enabled mobiles devices that could be infected with viruses wirelessly. To our dismay, the answer was: *a lot*.

In less than 23 hours, more than 1400 vulnerable devices were discovered, most of which were mobile phones. This number reflects the low level of public awareness about the potential dangers. Phone users also underestimate the possible damage if their mobile phones are compromised.

These kinds of security breaches also have serious consequences for corporations and telephone companies. However, most security teams in corporations simply believe that mobile phones are for individual use only. It is thus not their duty to protect these applications or even to understand the security problems. Hence, mobile phones may be the next easy target for professional hackers [24].

3.1 Overview of Bluetooth Technology

Bluetooth is named after the King Harald Bluetooth who united Denmark and Norway. He was known for his skill in getting people to talk to each other. This technology [25,26] enables two devices to communication wirelessly within a short distance. Such devices include computers, mobile phones, headsets, keyboards, printers, etc. The initial objective of Bluetooth was to replace cables with low-cost, low-power wireless technology. Its functions and standards were further developed by the Bluetooth Special Interest Group (SIG). Members of this group include Ericsson, Nokia, IBM, Microsoft, Intel, Toshiba, and other companies. The Bluetooth-enabled equipment shipments have grown to 800 million in 2007 and its predicted growth is over 1.8 billion by 2012 [27].

This technology can also be used to replace the inconvenient Infrared. It makes it easy to synchronize devices and transfer files. However, it is not a competitor of IEEE802.11 standards as the low-power Bluetooth can operate only within a

relatively short distance. It simply enables users to eliminate or minimize the number of cables between computers and peripherals.

Two Bluetooth-enabled devices can communicate with each other directly. For example, two mobile phones can exchange data in a basement that is not covered by a telephone company signal. Bluetooth uses a free 2.4 GHz radio band to communicate within a range of up to 10 or 100 m (depending on the model). Examples of Bluetooth applications include the following:

- A user can answer phone calls with a Bluetooth headset when the phone is in his/her pocket or briefcase. In some countries, it is illegal to hold a phone while one is driving. Bluetooth headsets/earphones provide a perfect solution for drivers.
- Computers can transmit data to printers without cable connections.
- PDA/smart phones can synchronize files with PCs.
- Business cards, photos, and ring tones can be exchanged between phones.

3.1.1 Improper Implementation

Bluetooth technology is designed with security issues in mind. Nothing is perfect, however, so there are weaknesses [25,28] in the standards. Most threats [29–31], though, come from improper implementation by manufacturers.

For example, one important Bluetooth security feature is the user's ability to switch between the ''discoverable'' and ''hidden'' modes. Every Bluetooth device should have a unique address that is 48 bits long. To connect to a Bluetooth device, this address must be known. It is extremely easy for a PC to scan the addresses of nearby Bluetooth devices if they are in the ''discoverable'' mode. Switching the device to ''hidden'' mode will provide much better protection from unauthorized connection.

The default setting of some mobile phones is the ''discoverable'' mode. Because most users do not understand Bluetooth technology, they do not switch their phones to the ''hidden'' mode. To make matters worse, it is difficult in some phones to find the right menu to turn the mode to ''hidden,'' because of poor HCI design. The wrong default setting exposes users to great risk.

Although the ''hidden'' mode is not a foolproof method of protection [23,28], it does mean it will take hackers much longer to hack the devices. One hacking approach is the brute-force discovery method. An example is the RedFang program developed by Ollie Whitehouse, which tries to connect to each Bluetooth device, one by one. As each address is 48 bits long, it seems impossible to test all of the combinations (i.e., 2^{48}), as this would take years. Unfortunately, some

manufacturers assign the Bluetooth addresses in a predictable way which can further weaken the security features [25].

The first 3 bytes (24 bits) are assigned to a specific manufacturer, which is free to allocate the last 3 bytes to its own products. In other words, the first 3 bytes of all of the products of a certain manufacturer will be identical. If hackers know the manufacturer of the phone, then the number of possible address will be 16,777,216 (i.e., 2^{24}). Unfortunately, some manufacturers assign the last 3 bytes in a predictable way. For example, a manufacturer might use the first byte (of the remaining 3 bytes) to identify a particular phone model. The addresses of all phones of the same model will start with the same 4 bytes, so hackers need to try only the last 2 bytes. The number of combinations then drops to 65,536 (i.e., 2^{16}). This example shows how improper implementation can weaken the security features. Although 65,536 is still a large number, hackers can use multiple computers (or a computer with multiple Bluetooth dongles) to carry out simultaneous attacks, or use other techniques to speed up the process. The model of a phone can be learned if it used in a public place once. Some people get used to putting the phones on a table or hanging them around their necks.

Another weakness is in the pairing process [25,28] that two Bluetooth devices need to go through before data exchange. Devices need to send an identical PIN to each other. A key will then be generated and stored in both devices so that later communication can take place automatically. However, the first step of this pairing process is done in plain text and is not encrypted. Hackers can intercept the communication during pairing. These pieces of information will help the hackers to speed up the hacking process later [25].

Improper implementation in the pairing process includes the use of very short PINs instead of longer and more secure PINs. Poor HCI design also deters users from changing the default PINs in certain Bluetooth-enabled devices.

It is also possible to force the devices to repeat the pairing for each round of communication. This provides the illusion of security, but, in actuality, it provides attackers with more opportunities to intercept the messages during the pairing process.

3.1.2 Proof-of-Concept Viruses and Tools

Security researchers have developed viruses and tools to prove the feasibility of attacks on mobile phones. We introduce several of them, so that readers will have a better idea of the state of the art in phone virus development:

- *Cabir*. This is the first Bluetooth virus, and was released in 2003. It can infect certain makes of phones with Bluetooth technology. The infected phone will

replicate itself quickly to other phones through Bluetooth communications. This virus is not dangerous, as it only consumes battery power and does not cause any other serious damage.

- *Commwarrior*. In addition to spreading among Bluetooth devices, this virus will also read users' phones and send MMS messages containing the Commwarrior file. This virus can replicate itself quickly all over the world, as it is not restricted by the short distances of Bluetooth technology.
- *Blooover*. The BlueBag project (mentioned at the beginning of this chapter) needs a computer to carry out attacks. The Blooover (from the phrase ''Bluetooth Hoover'') tool proves that it is also possible to use a smart phone to perform similar attacks. It is written in Java and runs on a smart phone. Blooover can detect vulnerable phones and carry out other attacks. It can also initiate phone calls and perform call forwarding.
- *BlueSniper ''rifle.''* Some phone manufacturers underestimate the threats against Bluetooth, because they believe Bluetooth hacking can only be done over a short distance. John Hering and his colleagues at Flexilis developed a device in the shape of a rifle [31]. Using a vision scope and an antenna that connects to a laptop or PDA, they aimed the rifle at a taxi stand from the 11th floor of a hotel in Las Vegas. They were able to collect 300 address books from Bluetooth-enabled devices. In later experiments it was successfully used on a target 1.1 miles away. This shows that the distance can be extended greatly with the right equipment. It also proves the possibility of picking out a particular victim from among a group.

3.2 Illusions

Manufacturers incorporate PDA features into mobile phones to make them ''smart.'' Indeed, we can consider smart phones to be small computers with the full functions of phones. Users can read and send e-mail messages, and they can browse the Internet, read files, and work on spreadsheet applications. Smart phones are much lighter and smaller than notebooks, making them good tools for business. The number of smart phones worldwide will reach 500 million by 2012 [32].

As smart phones can perform the tasks of a computer, they are vulnerable to the same kind of hacking attacks. However, there are many misunderstandings, and most mobile phone owners tend to believe:

- There are enough security features in the phones.
- Hackers are not interested in phones.
- There is no such thing as a phone virus.
- They have little to lose even if their phones are hacked.

3.3 Reasons for the Misunderstandings

Many high-profile computer hacking cases have been widely discussed in newspapers and on television. Yet the hacking of mobile phones is hardly ever mentioned in the media. Users might believe that the existence of phone viruses is only a rumor, as they are not aware of the new threats.

In the past, hackers were teenagers and amateurs who wanted to prove their talent by paralyzing large numbers of computers. Their work attracted a lot of attention, but they rarely gained any financial benefit. However, the new trend in hacking is that it has become a full-time job. Hackers quietly steal information from victims' computers and gain financial rewards. Victims might not discover these intrusions for a long time, and some of them might never discover the heists. These kinds of hackers do not want any publicity, as it would increase their chances of being caught. This creates the illusion that hacking activities are much rarer than they are in reality.

Furthermore, many corporations are unwilling to reveal security breaches in their organizations, particularly when these events involve the sensitive data of clients. Not only would such revelations be public relations disasters, but also they would also cause further financial losses. Admitting negligence could invite lawsuits from clients and law enforcement agents [33]. No news is not good news in the world of computer security.

3.4 Possible Attacks

All hacking problems related to computers could happen via smart phones. Furthermore, there are unique problems, as phones have more functions. Users also tend to store more personal data in phones. Examples of these problems include the following:

- The ability of a phone to call certain numbers, such as the emergency number 911, could be blocked.
- Hackers could retrieve address books, calendars, photos, or other files from a phone.
- Compromised phones could infect other phones that use Bluetooth or MMS.
- Hackers could send SMS or MMS to other phones without any user interaction.
- Hackers could remotely control a phone to make phone calls or connect to the Internet.

3.5 Consequences

These problems seem to be harmless, so most users do not recognize the serious consequences. Some possible consequences will now be discussed:

- *Leaking calendars and address books*. Hackers could sell these pieces of information to a user's competitors. This is valuable information, as rivals could find the names of clients (or potential clients). Hackers could also alter the details of a user's calendar. As a result, the user could miss important appointments with his/her clients, while rivals approach them with another proposal. Losing important clients could be detrimental to the user and his/her employer.
- *Bugging devices*. Hackers could instruct the user's phone to make a phone call to the attacker's phone without the user's consent. They can then eavesdrop on (or even record) the user's conversation and the phone then becomes a potent bugging device.

  The user might discover these calls in the phone's call registry. However, he/she may believe that the calls were initiated due to the accidental pushing of certain buttons when the phone was in a pocket or bag. More advanced hackers can erase the records on the phone. Prudent hackers can even use prepaid phone cards, so that it is impossible to trace their identities afterwards.

  It is also possible for hackers to set up a gateway, so that users' phone calls are routed through the hackers' phones so they can listen to users' phone communications.
- *Sending SMS*. Terrorists can use SMS messages. For example, they could send false bomb threats to airlines using legitimate users' phones. This would consume government resources as the government investigates false leads while the terrorists carry out the real attacks. There would be no way to trace the terrorists, and the phone owners could be in serious trouble.
- *Causing financial losses*. Hackers could send a large number of MMS messages with a user's phone. MMS services are still quite expensive for large files, and it would be a burden for users to pay these phone bills. Updating or downloading large files with GPRS services would have same effect.
- *Revealing passwords*. People have many account names and passwords to remember these days. For example, they may have an e-mail account, a corporate account, an Internet banking account, supermarket accounts, etc. Some may also need to remember their ATM PINs, the code to open the gate of their apartment building, and another code to activate the alarm systems of their office or home. As mobile phone users almost always carry their phones with them, they may believe them to be a safe and convenient place to store

these account numbers and passwords. Attackers, however, can retrieve such passwords on compromised phones. The disclosure of these pieces of information not only endangers the phone users themselves, but also jeopardizes the security of their employers' security systems.

- *Identity theft*. There are black markets in which hackers can buy and sell personal information [24,34]. The price varies depending on the sensitivity of the data and the value of the victims. For example, Jeremy Jaynes in the US managed to earn $24 million before he was arrested [3]. His case involved a stolen database that included a million personal detail records.
- *Attacks on telephone networks*. If a virus infects a large number of phones, it can instruct all of them to make phone calls (or send SMS or MMS messages) simultaneously at a certain time. This tactic would paralyze a city's telephone networks and create chaos.
- *Leaking corporation data*. Employees can download files from the company's computer onto their phones so that they can continue to work at home. Many high-end mobile phones can also receive e-mail messages. They enable users to keep in touch with their colleagues and clients. It would be a disaster if the phones were hacked.

3.6 Phone Virus Development

Developing software for phones is more difficult and complicated than it is for ordinary computers. As a result, the development usually takes much longer. Development difficulties include the following:

- The memory of a phone is much smaller than that of a computer, so programs must be carefully designed to reduce the size. Special tools are usually required for development. For example, Java programmers need to use J2ME instead of J2EE, which is used for computers. It takes a while for a computer programmer to pick up the skills needed for phone development.
- The screen is quite small, so application messages must be succinct.
- It is more time consuming and difficult for users to type in data. Phones usually have only 10–20 buttons, whereas a normal computer keyboard has more than 100. A good application design must minimize the input keystrokes for users.
- Phones come with many different standards. These include:
  - The size of the screen and resolutions
  - Memory size
  - The number of buttons (and/or interface methods, such as touch-screen, joystick, etc.)

  - Operating systems (e.g., Symbian, Linux, Palm, Microsoft, Android, etc.)
  - Programming languages
- Phones need to use the services of telephone companies in order to function. Unfortunately, there are many different standards for these services too.

Due to the existence of different standards, software developers must convert and test their programs for different brands and models. It is a nightmare for maintenance, too, as there are many versions of the same program for different platforms.

An efficient virus does not need any human interface, so small screens and different buttons on phones do not bother hackers. However, the other problems discussed above do slow down the development of virus programs. Hackers have learning curves too, which is why the number of phone viruses is relatively small. However, we will have a very different scenario in the near future for the following reasons:

- Phones are now being mass-produced in even greater numbers. Manufacturers are able to make phones with faster CPUs and more memory while maintaining the same price ranges. For example, the latest models come with 32 GB of storage. Phone viruses may become more sophisticated and perform subtler attacks.
- Software developers are aware of the above problems. There are now more tools for development, testing, and conversion between different models. It also takes less time to write programs than it did earlier. Ironically, virus writers also benefit from this progress.
- The number of business applications and phone users is growing every day. Hackers now have more to gain financially from attacking phones.
- It is easier to cheat phone users than computer users with social engineering techniques [35]. The media seldom discuss these kinds of dangers, and phone manufacturers also downplay the possible risks. Low levels of awareness among users make them easy targets. They get used to downloading ring tones, wallpaper, games, etc.

  Viruses can masquerade as operating system updates or telephone company services. Due to small phone screens, very little information about the source and the received file will be displayed. It is very difficult for users to judge whether a file comes from a trusted source. The success rate of hacking will thus be higher, and more attackers will switch to hacking phones as it becomes more cost effective for them.

3.7 Countermeasures

Countermeasures will not be effective without cooperation between wireless device manufacturers, corporations, and users. One party's actions will change the behavior of the others. We present their responsibilities in the following sections.

3.7.1 Manufacturer Responsibilities

Some manufacturers claim that mobile phones are safe from hacking. Their strongest arguments are that attacks must be carried out over short distances and that hackers do not have enough time to hack the phones.

Unfortunately, this is not always true, as is clear in the following examples:

- The communication distance of Bluetooth devices can be easily extended with inexpensive antenna, as in the BlueSniper experiments.
- A user might go to the same restaurant/fast food chain for breakfast or lunch every day. An assiduous hacker has plenty of time to carry out an attack.
- An attacker might be a user's colleague [36]. There are at least 8 h every day over a prolonged period in which to make the attack.
- Bluetooth is not the only method of spreading viruses. Users with MMS, GPRS, or ''WAP push'' might be convinced to install a virus through social engineering technique. Once a phone is infected, the virus can replicate itself in the phones of colleagues or family members. In some countries, mobile telephone services are less expensive than fixed lines. Many people never turn off their phones because they simply do not have fixed telephone lines at home. Family members' phones have months and even years to infect each other.

New specification, Bluetooth version 3, has been finalized to address the above weaknesses in April 2009. However, it will take at least another year until devices supporting the new Bluetooth specification are on the market. Till then, all old devices are vulnerable to the existing attacks. Furthermore, hackers will always find new ways to attack. However, some measures should be adopted to make devices safer and more difficult to attack. These measures include the following:

- Manufacturers should allocate more resources to implement security properly.
- Logs of bluetooth pairing activities should be maintained so that users can detect any intrusion attempts as soon as possible.
- The default security setting should be at the maximum level.
- Manufacturers should pay attention to HCI design so that users can fine-tune the security settings easily.

- Security issues should be discussed in the user's manual.
- Usually, the easiest way to fix a security flaw is to update the phones' firmware. Manufacturers should inform registered users as soon as possible when new firmware is available.

3.7.2 Corporation Responsibilities

Phones viruses will hit corporations and cause substantial damage in the future. It is only a matter of time. Equipped with powerful smart phones, employees will discover new ways to improve their jobs. For example, they can download files from the company's computer onto their phones so that they can continue to work at home. This is better than using USB drives, as employees do not even need to take the phones out of their pockets or briefcases. When they go home, they may forget to pull the USB drives from their computers, but they seldom leave their phones in the office. However, it would be a disaster if the phones were lost or hacked. Security teams need to be prepared for future challenges. Corporations should take a proactive approach. Examples of the countermeasures they could take include the following:

- Devise a policy for the use of mobile devices by their employees.
- Learn and monitor the latest developments in phones and other related technology.
- Employees do not have the knowledge to choose phones with good security features, because there are so many models. Corporations should provide a list of phones with good security features and HCI design. If enough corporations do the same thing, it will force manufacturers to spend more resources on implementing security features correctly.
- Educate employees to select and use their phones properly. They should be taught to fine-tune the security features. For example, a short seminar could be offered to employees, and up-to-date guidelines could be issued. The content of these guidelines may change over time, as technology and hacking techniques will continue to advance. The guidelines could include the following:
  - Switch the Bluetooth security setting to ''hidden'' mode.
  - Activate Bluetooth only when it is needed.
  - Do not accept any unsolicited pairing requests.
  - Minimize pairing operations in public areas.
  - Monitor the numbers and names of the paired devices on the phone to discover any suspicious connections.

- Update the phone's firmware when there is a new version available.
- Pay attention if the phone consumes power at a faster rate than is normal or there are any other anomalies.

3.7.3 User Responsibilities

It is the duty of phone users to protect themselves and their employers' data. In addition to adhering to the above guidelines, it would be useful for users to do the following:

- Be vigilant and treat a smart phone as a computer. Almost all of the precautionary measures for computers can be applied to phones.
- Invest in the time to update their knowledge, for example, by attending employers' seminars.
- Be aware of social engineering techniques.
- Check the following logs on the phone, if available:
  - GPRS usages
  - ''Phone call'' records
  - If the usages show any dramatic increase, the user should try to identify the problems with assistance of experts.
- Verify the sender of suspicious SMS, MMS, and e-mailing messages on the phone.
- Encrypt data files on the phones.
- Install antivirus software package for mobile phones.

4. Threats of Public Wireless Networks

Wi-Fi is a more consumer friendly term referring to wireless networks which use the IEEE802.11 family wireless networking protocols (discussed in Section 2.1.1). Many telephone service companies, Internet services providers, governments, and commercial firms set up Wi-Fi networks in public places such as airports, train stations, bus stations, coffee shops, etc.

Users can, for example, check their e-mail messages, surf the Internet, conduct e-banking transactions, or access the databases of their employer in a public place, as long as it is covered by these Wi-Fi networks. ''Hot spots'' are the places where people can receive the Wi-Fi signals.

Some Wi-Fi networks are free of charge. For example, some coffee shops provide free wireless connections for their clients. Some local or national governments install free networks in order to promote a better image and/or attract tourists. Some Wi-Fi networks require the users to subscribe to the service before they can use networks. Many users believe these networks are safe. However, this is not true as many wireless network operators fail to implement enough security features.

In the early stage of public Wi-Fi networks, users usually accessed the networks with notebook computers. Nowadays, many high-end smart phones are Wi-Fi enabled. There are two ways for owners of such smart phones to surf the Internet:

- Use the telephone signal (e.g., GSM) to access the Internet (Fig. 8).
- Use the Wi-Fi networks (Fig. 9).

The telephone signals are encrypted so the communications are protected. Although this encryption can be cracked, attackers need expensive facilities. It is also more difficult to obtain these facilities. At the time of writing, these facilities are usually used by law enforcement agents and not affordable to ordinary attackers.

Instead of using the telephone signals to surf the Internet, smart phone owners can use the Wi-Fi signals. Doing this provides them the following advantages:

- The Wi-Fi speed is usually higher.
- The charges (if any) for Wi-Fi connection is usually lower.
- Some places (e.g., inside a mall with thick walls) are not covered by telephone signals.

On the other hand, the Wi-Fi networks are easier to crack if enough security features are not properly implemented. Smart phones can automatically connected to unsecured Wi-Fi networks. It is quite difficult for the owners to distinguish whether they are accessing the Internet with Wi-Fi or telephone signals due to poor HCI design. Owners might not be able to detect that their phones switch the connection methods in a particular places. Thus they do not pay attention to security.

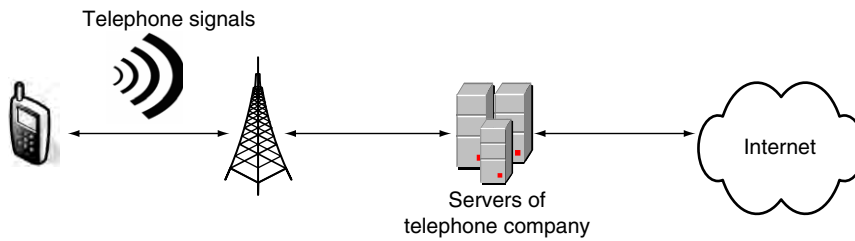

FIG. 8. Using telephone signals.

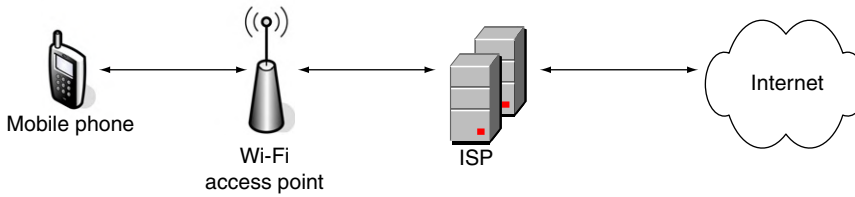

FIG. 9. Using Wi-Fi.

4.1 War Driving

''War Driving'' is the act for searching both public and private wireless networks around a specific area, mapping the population of wireless access points for statistical purposes. The first officially recognized War Driving was performed by Peter Shipley in 1999, who presented his work to the hacker community in 2001 [10]. Since then the practice has been conducted in order to gather data to promote awareness of wireless technology applications.

These statistics can be used to raise awareness of the security problems with wireless networks [37]. The person who performs War Driving usually takes a moving vehicle with a wireless equipped computer, such as a notebook computer, PDA or smart phones. However, if someone performs War Driving for the purposes stealing Internet access or committing computer crimes, the activity would be treated as a criminal offence [38].

4.1.1 Tools for Performing War Driving

Wi-Fi equipped computers can use different software packages to perform War Driving. Wireless security auditor (WSA), a complete wireless network analysis and auditing program, helps normal users close any vulnerability before hackers try to break in. Other software packages, such as AirMagnet, Sniffer PDA, and Fluke Wave Runner, are intended for expert protocol users, and allow the capture of wireless packets for detailed analysis [39]. Most of these packages can be obtained from the Internet at no cost.

4.1.2 Survey Results

We summarized the latest War Driving results in Tables A.I–A.III. Although there is a trend that networks are becoming more secure each year, there are still a large number of insecure networks. Some experiments have also been conducted

within commercial areas. The results indicate that ignorance is not limited to individual users. Insecure networks are also common in many corporations. This reflects the low level of awareness in both individuals and corporations.

4.2 Possible Attacks

Wireless networks have almost all the security issues of wired networks. Wireless networks provide many advantages. However, wireless networks cannot replace the wired connection. Thus both wireless networks and wired networks will continue to coexist in the foreseeable future. Many messages need to travel through both wired and wireless networks before they reach their final destinations. In addition to the old issues, there are some new security threats which are unique to wireless networks due to the following reasons:

- It is very easy to capture the wireless communications as attackers do not need to tap to the cables. Even if the wireless communications are encrypted, the attacker can record the messages and do the analysis at a later time.
- It is more difficult to identify the attackers.
- Attackers need only very simple facilities.

We can consider wireless threats as a superset of the wired problem, as depicted in Fig. 10.

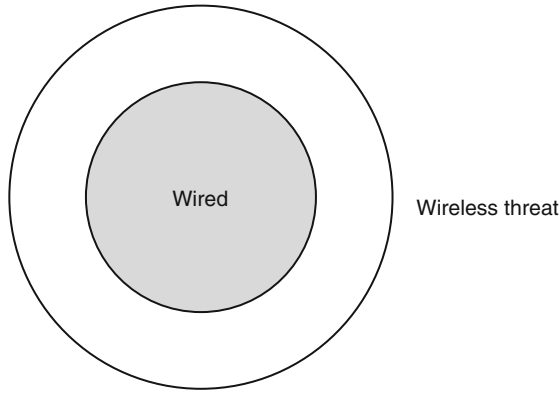

FIG. 10. Threats of wired and wireless networks.

4.2.1 Bogus Access Point

Attackers can set up bogus access points (Fig. 11) in public places. Bogus access points can have the same SSID of any popular service provider. Once connected, the bogus access point can direct the user to legitimate servers. The user might believe his communications are protected by the service provider. However, all communications go through the access point so the attacker can intercept the information easily.

On the other hand, the bogus access point can also direct the user to the attacker's server. For example, the attacker's server can masquerade as the user's bank. It can ask the user to supply the account number, password, and other sensitive information.

4.2.2 Jamming

Jamming can break communication between users and the legitimate access point. The attacker listens to the communication first and collects information on the user. Next he can break the communication at the right moment by sending strong wireless signals to the user. He can then impersonate the jammed user to continue communication as in Fig. 12. The access point and server cannot detect this kind of interruption if there are insufficient security measures.

It is also possible for an attacker to install an access point and impersonate the server as shown in Fig. 13. Access points can be purchased at a low cost and easily

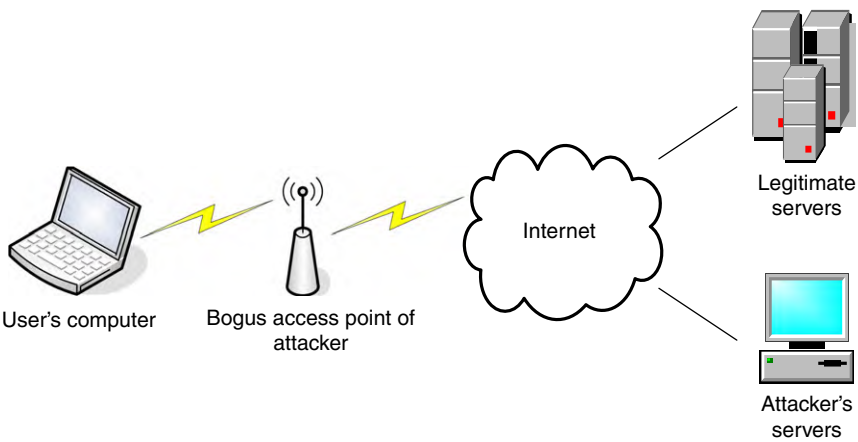

FIG. 11. Bogus access point.

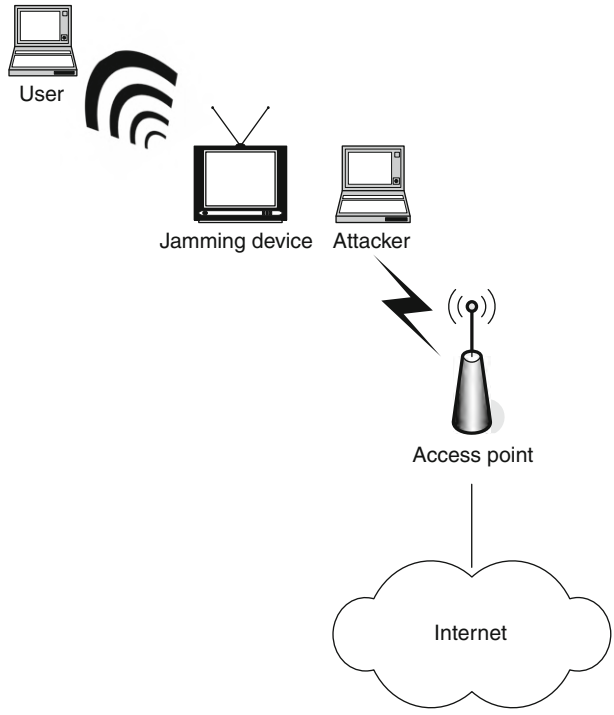

FIG. 12. User jammed by attacker.

configured with limited knowledge. Indeed the rogue access point needs only a stronger signal than the existing access point in order to intercept the user signals. When the user connects to the attacker's computer, the attacker can ask the user to supply his/her password and/or other important information. The attacker can then use the account and password to attack the network. Similar masquerading attacks are much more difficult in a wired network. Many vendors of wireless access points are not providing sufficient security features to protect networks from such attacks.

4.2.3 Man-in-the-Middle Attack

A Man-in-the-Middle (MITM) attack intercepts the message from the network. The attacker then modifies and/or adds data in the message. This modified message is injected back into the network to continue its journey. If the attack is successful,

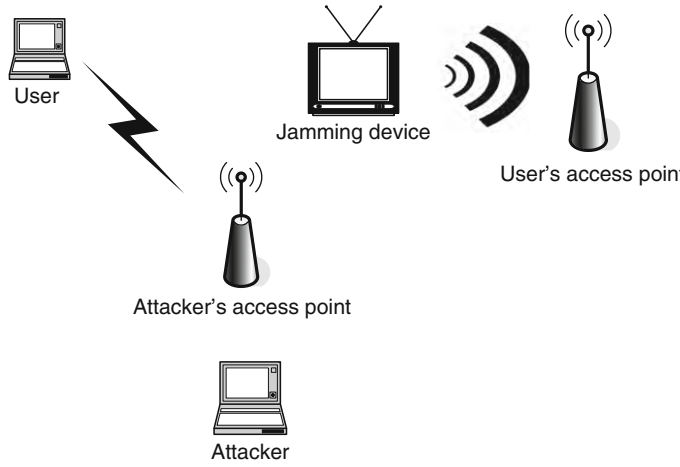

FIG. 13. Access point jammed by attacker.

the attacker can impersonate either the user or the server. Such attacks can create both confidentiality and integrity problems.

This process is more complex than other attacks. The attacker must have good communications knowledge and also needs to collect information about the network before he can launch his attack. However, there are some software packages designed for this task and they are available in the public domain. An attacker can use these packages to gain unauthorized access to networks (Fig. 14).

5. Illusions of Encryptions

Let us examine the process of sending a letter from a small town of country A to another small town of country B. The closest post office will be the one which sorts and distributes it to a central office. This central post office might send it to another office with an international airport in country A. The letter will go to the post office of a big city in country B with its own international airport. The letter will then go through similar sorting and distribution processes in different post offices before it reaches its destination. Similarly, any Internet communication messages must be relayed by many servers before they reach their destination. The situation is even worse, in fact: there are no physical envelopes. People can read the messages easily unless their proper protection is put in place.

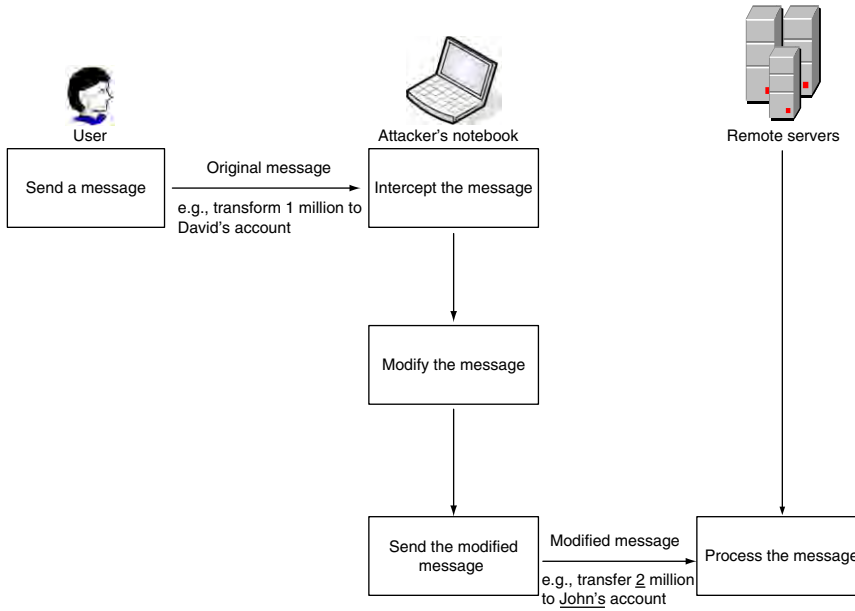

FIG. 14. Man-in-the-Middle attack.

Encryption technologies can solve a lot of problems mentioned. Ironically, they can also cause illusions. Users might believe that everything is protected once they use encryption techniques. However, modern computer systems consist of many different parts and functions. There are many different encryption techniques. Most of the time, one encryption technique can provide protection to some parts only. In the rest of this section, we will present some basic concepts of encryptions. We will then discuss what is actually protected.

5.1 Basic Concept of Encryption

To put it simply, encryption is the process of converting original data into unreadable code. It can then be transmitted to the destination. Once it reaches the destination, it can be transformed back into the original message. Even if it is intercepted by a third party, the attacker cannot understand the meaning of the code. Nowadays, encryption techniques can also be used to authenticate users or servers, and to verify the integrity of data.

5.1.1 Single Key Encryptions

Single key (or secret key) encryption consists of two parts—a key and an algorithm. An algorithm is a series of complex operations. There are over thousand encryption algorithms. Many security experts are still trying to develop more efficient algorithms today. Discussion of the details of these algorithms is beyond the scope of this chapter. Interested readers should refer to Refs [40–42]. We will only present a simple example so readers can understand the later part of our discussion.

For example, party A wants to send a message ''dog'' to party B. Party A can use the following technique:

- Key: 2
- Operation (algorithm): shift each character to two positions according to the alphabetical list as in Fig. 15.

After the transformation, party A will send ''fqi'' to party B. Party B can convert it back to ''dog'' if he has the key ''2'' and knows the operation. Of course this encryption can be ''cracked'' within seconds with the aid of computers. In real life, the key is usually a long string of characters and the operations are more complex. It will be very time consuming to crack the encryption. This technique is also called ''symmetric key'' as both parties have the same key for encryption and decryption.

The single key system has the following problems:

- A key must be sent from one party to another before they can communicate with each other with encryption. The key itself must be sent in a secure way. Sometimes this is inconvenient or even impossible.
- Key management can be a time-consuming exercise. For example, party A wants to talk with party B and C. He needs to keep two keys if he does not want B to understand the conversation between A and C. If someone wants to communicate with 100 parties, he needs 100 keys. It will be a very heavy burden for ordinary users.

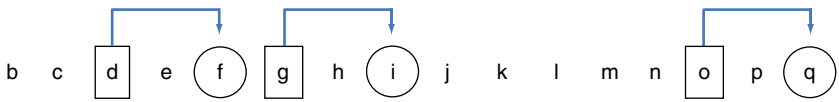

FIG. 15. Shift each character to two positions.

5.1.2 Two Keys (Public Key) Encryptions

In the two keys system, user has two keys—one private key and one public key. He can post his public key on a Web page or send it to his friends through unsecured e-mail messages. However, he must keep the private key secret.

For example, party B uses the public key of party A to encrypt the message and send it to party A. Party A can then decrypt the incoming message with his private key as in Fig. 16.

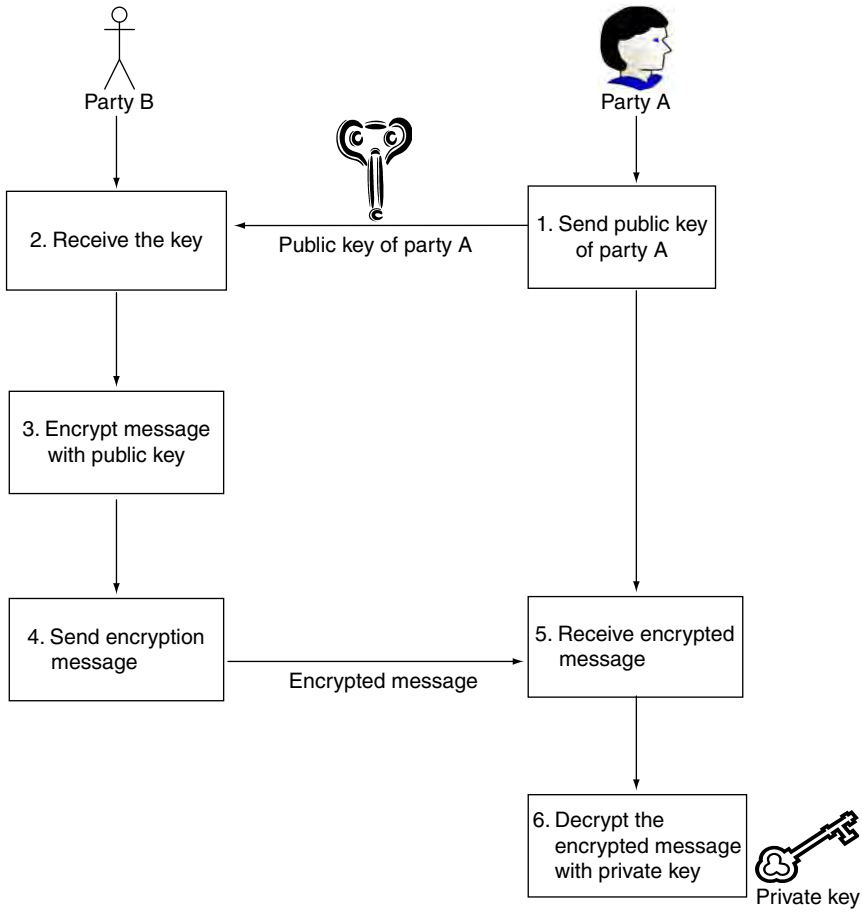

FIG. 16. Public key and private key.

The operations are similar to the following translation example. The user has two dictionaries (i.e., an English-African and African-English dictionary). We are referring to a particular African language which is used only by a very small and extinct tribe. He can distribute the English-African dictionary to all his friends. A friend can use this dictionary to translate an English message to an African message easily. His friend can send this message to the user through a unsecured network. The user can translate it back to English as he has an African-English message (i.e., a dictionary which is arranged in the sequence of African words).

An attacker cannot translate the African message quickly if he has only an English-African dictionary. He needs to search the whole dictionary in order to find one African word because it is arranged in the alphabetical order of English. Thus it will take a much longer time for the attacker to crack the message.

In theory, most encryptions can be cracked. However, it will be useless for the attacker if he takes 100 years to crack it with the fastest computer. It will not be a threat to the sender and recipient as both of them will have long passed away! Public and private key encryption techniques solve the key management problem. However, single key is usually more computationally efficient. A lot of modern systems use both techniques.

5.2 Other Applications of Encryption

In addition to encrypting data, this technique can provide the following functions:

1. Authentication
    - It can verify the identity of a party. This could be either an individual or an organization.
    - Traditional authentication depends on account name and password to authenticate its users. However, it is not safe enough for critical systems, such as banking systems, for the following reasons:
        - Users do not choose their passwords wisely (e.g., their birthdays).
        - Users do not protect their passwords properly. For example, they might write them down and stick them on their computer monitors.
    - Basically it is a two keys system (Fig. 17). The server sends a challenge (usually a string of characters). The user's computer generates a message (called response) using his private key and ''challenge.'' The response is sent to the server. The server then verifies the response with another key. The challenge/response is different every time. An attacker will not be able to crack the system even if he can intercept them.

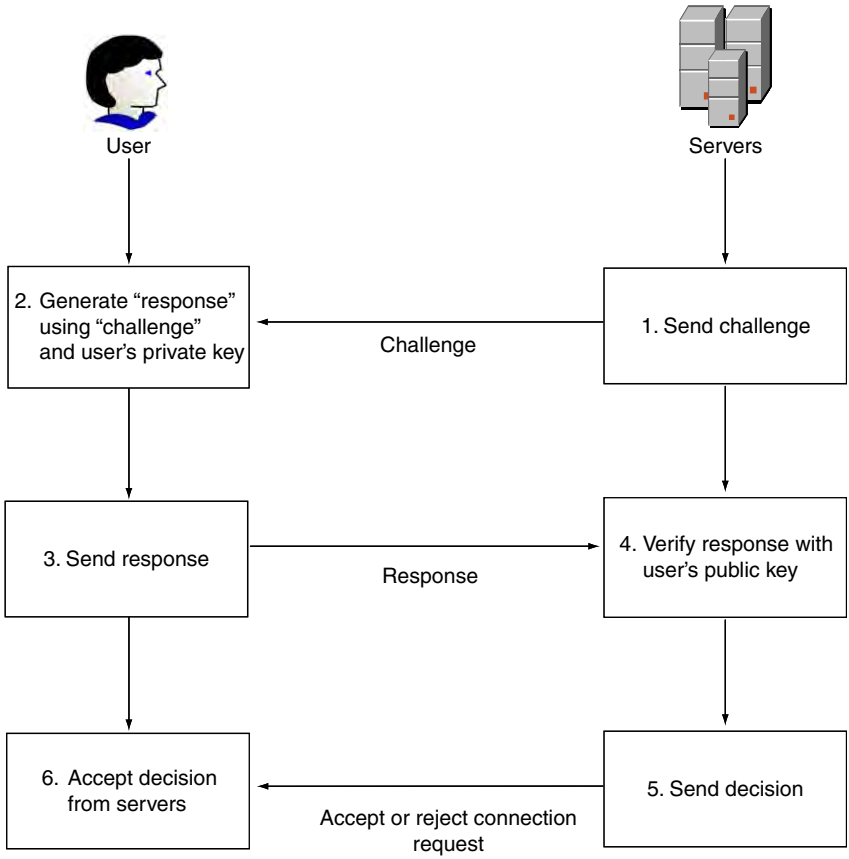

FIG. 17. Challenge and response.

2. Digital signatures
   - The digital signature technique produces a digital fingerprint which is unique to a message. It verifies that a message has not been altered in transit.

5.3 Illusions

As discussed earlier, encryption can generate illusions to users because they believe everything is safe. In many encryption methods, users need to select keys for the encryption. Weak keys can be cracked quickly and they cannot provide real protections. Guidelines for good keys/password management include:

- Do not use a complete word which can be found in a dictionary.
- Do not use the names of your spouse, children, pets, yourself, etc.
- Do not use addresses or telephone numbers.
- Do not write down the keys/passwords.
- Do not reuse keys/passwords for different accounts/devices.
- It should be at least eight characters long.
- It should consist of at least one number, one alphabetical character, and one special symbol (e.g., #, $, etc.).
- Keys/passwords should be changed periodically.

In addition to password/keys selection, we will examine other illusions one by one.

5.3.1 Encryption in Routers

Turning on the encryption in the routers only protects the communication between the user's computer and the router (Fig. 18). Users' messages will still go through the Internet which is unprotected.

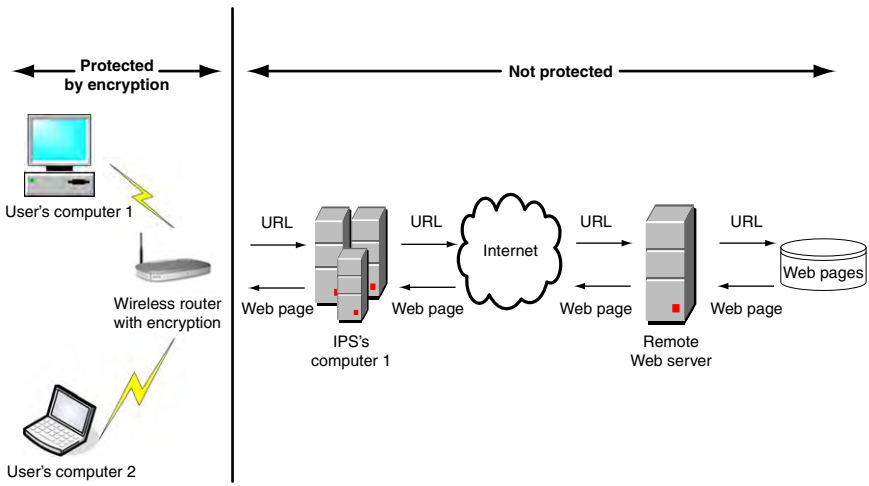

FIG. 18. Encryption in the router.

5.3.2 Web Server with Secure Socket Layer

Some Web servers use SSL which is basically a public key technique to encrypt messages (Fig. 19). Most Internet browsers such as Internet Explorer or Firefox can handle key exchanges and encryptions between the user's computers and Web servers automatically. Users do not need to do anything to initiate the encryption. A particular sign (such as a lock) will be displayed on the certain position of the Web page. However, a server might not protect all communications. It might only protect a certain sensitive page such as login pages. Users need to watch the special sign on every page if he is working on important information. Some companies also have several servers. Users might be interacting with different servers at different times. The servers might implement different levels of security.

5.3.3 Virtual Public Network

An employee can access his e-mail or company account in a hot spot if he has a notebook computer or smart phone. However, because the communication needs to go through a public network, information can be intercepted by the attacker in the middle. SSL protects only the communication with the Internet browser as discussed in Section 5.3.2.

Virtual public network (VPN) is another solution which protects the communication between two computers. To implement a VPN, a client program needs to be executed in the user's computer. Data packets between the user's computer and the corporation's communication will be encrypted.

VPN technology uses the idea of tunneling. VPN tunneling establishes a logical network connection (Fig. 20). Such connection may include nodes in a public network. Transmissions between VPN client and server are protected. The security level is as good as using a private network. That is why we call it *virtual private network*. However, communications with other servers will not be protected.

5.3.4 Digital Certificate

Digital certificate is an electronic file which can be used to verify the identity of a party on the Internet. We can consider it an ''electronic passport.'' Both individuals and corporations can be identified by checking the digital certificate. We trust a passport because we trust that the issuing body (i.e., the government in this case) does a good job in identifying the passport holder. However, there are some governments with corrupted officers or poor efficiency. The strength of identification depends on the trustworthiness of the issuing government.

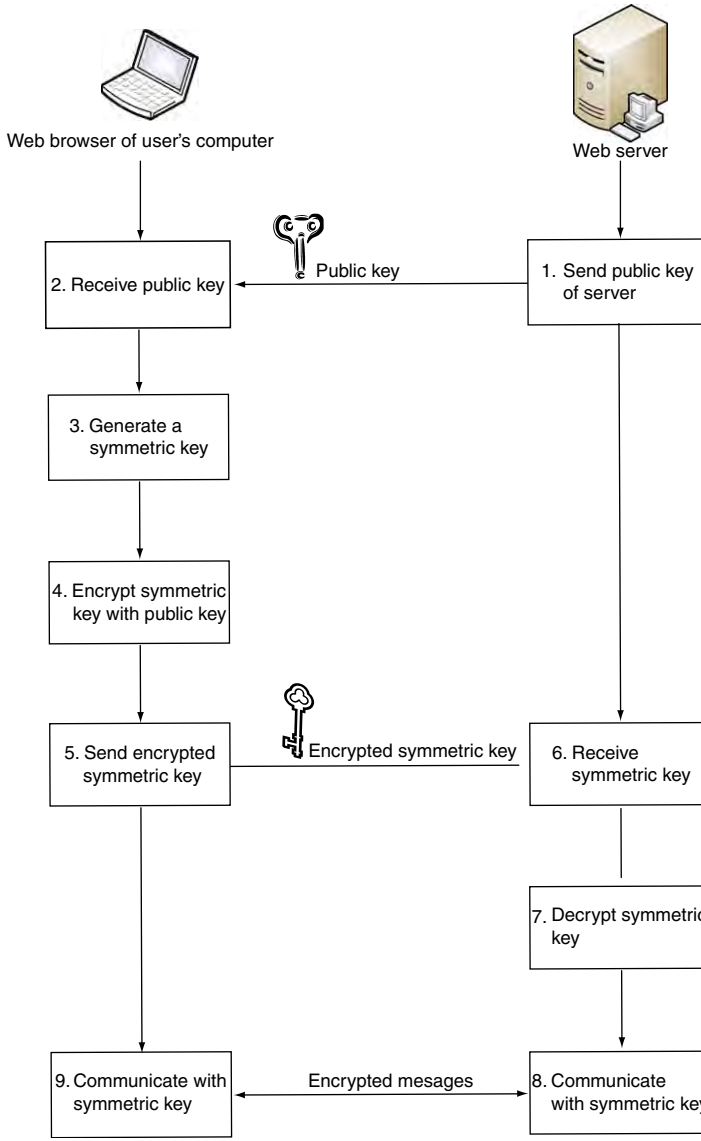

FIG. 19. Encryption with SSL.

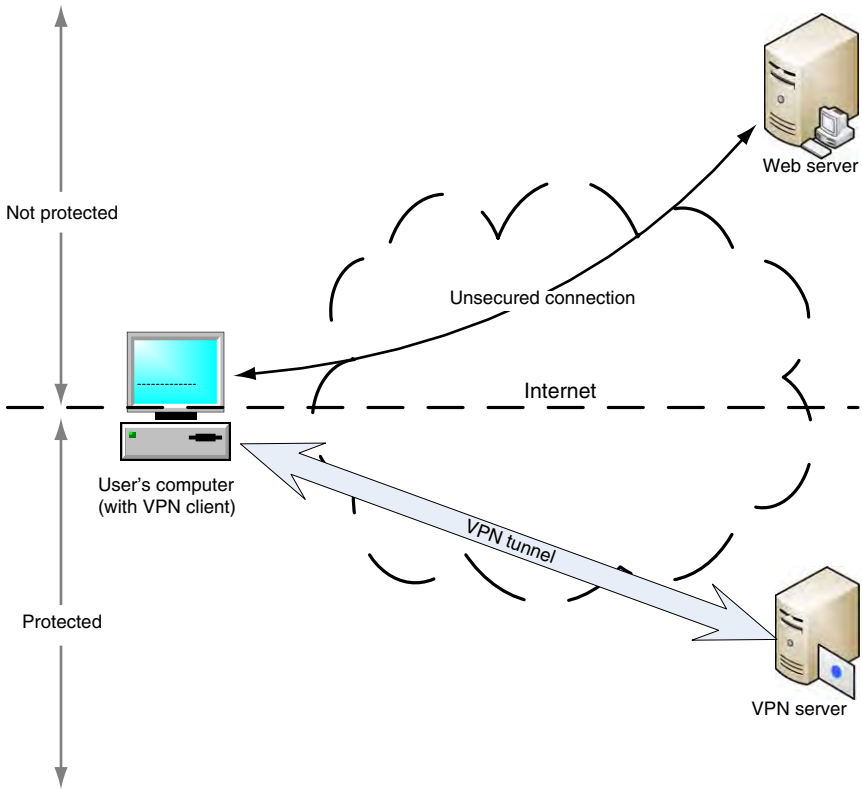

FIG. 20. VPN tunnel.

Digital certificate is similar to the passport case. It is issued by an organization called a certificate authority (CA). However, there are many CAs which can issue the digital certificates. An attacker can also issue a certificate to himself. In addition to checking the digital certificate, users must check the issuing body. Unfortunately users have very limited knowledge on the trustworthiness of CAs. Corporations should protect a list of trusted CAs to their users.

6. Conclusion

Technology alone will never be able to solve all security problems. Enhancement of the coordination between employers, end-users, and wireless facilities manufacturers is constantly required. Users should understand that it is their obligation to

protect their employers' computer systems by understanding the risks and appropriate countermeasures, and that it would be worthwhile investing their time in updating their knowledge. Employers and manufacturers should make this process as easy as possible. Indeed, common sense, constant vigilance, and up-to-date knowledge are the best weapons in the fight against hacking.

It is also important for corporations to set up a clear user policy. In addition to establishing the policy, they also need to review it frequently and enforce it effectively.

Every technology has its weaknesses. The risks of using wireless technology are relatively low compared with other technologies, provided they are used properly. Most of the existing threats come from the ignorance of users, improper security implementations by some manufacturers, and the inactive attitude of many corporations. Better coordination among these three parties, up-to-date knowledge, and vigilance will substantially reduce the risks.

There is no silver bullet or panacea in the fight against hacking. However, it is interesting to note the sad, but true, ''Tiger'' theory: ''to survive in the jungle, one does not need to run faster than the tiger. All one needs to do is to run faster than the other people. The tiger is not interested in chasing the fastest runner.'' If a corporation has a reasonable level of security measures, rational hackers will attack other, weaker organizations, because their hacking will be more cost effective. It always pays to be the leader in the implementation of proper security measures.

ACKNOWLEDGMENTS

This work in part has been supported by the research funding of Lingnan University. Parts of this chapter have been published in the following papers (by the same author):

- A. Loo, Security threats of smart phones and Bluetooth, Commun. ACM 52 (3) (2009) (© ACM, 2009).
- A. Loo, The myths and truths of wireless security: an end-user perspective, Commun. ACM 51 (2) (2008) 66–71 (© ACM, 2008).

Appendix A

TABLE A.I
WI-FI SECURITY (WAR DRIVING) SURVEY IN 2006

	Cities			
Consideration factors of Wi-Fi security	Hong Kong	London	Paris	Hannover
WEP/WPA encryption enabled (%)	61.80	60.00	70.50	45.00
Adoption of WEP encryption (%)	79.31	N/A	N/A	89.00
Adoption of WPA/WPA2 encryption (%)	20.69	N/A	N/A	11.00
Adoption of factory default SSID (%)	44.33	4.00	1.39	0.60
Hiding of SSID (%)	N/A	30.00	33.00	8.00
Adoption of 802.11g (%)	82.77	67.60	85.00	51.20
Sample size	4344	Over 260	1000	300

TABLE A.II
WI-FI SECURITY (WAR DRIVING) SURVEY IN 2007

	Cities			
Consideration factors of Wi-Fi security	Hong Kong	London	Macau	Paris
WEP/WPA encryption enabled (%)	72.43	65.00	65.12	90.00
Adoption of WEP encryption (%)	72.49	N/A	85.12	41.00
Adoption of WPA/WPA2 encryption (%)	27.51	N/A	14.88	49.10
Adoption of factory default SSID (%)	30.00	1.50	44.36	5
Hiding of SSID (%)	N/A	19.40	N/A	N/A
Adoption of 802.11g AP (%)	91.00	81.90	89.36	N/A
Sample size	6662	Over 400	2923	18,313

TABLE A.III
WI-FI SECURITY (WAR DRIVING) SURVEY IN 2008

	Cities			
Consideration factors of Wi-Fi security	Hong Kong	Xian	Macau	Santiago
WEP/WPA encryption enabled (%)	78	53.90	77.00	84.00
Adoption of WEP encryption (%)	47	37.90	51.00	61.00
Adoption of WPA/WPA2 encryption (%)	31	16.00	26.00	23.00
Adoption of factory default SSID (%)	32	N/A	N/A	N/A
Hiding of SSID (%)	8	3.00	N/A	77.00
Adoption of 802.11g (%)	97	N/A	N/A	N/A
Sample size	7388	2015	Nil	Over 1700

Appendix B

TABLE B.I
DIFFERENCES OF DIFFERENT IEEE802.11 STANDARDS

	IEEE Version				
Features	IEEE802.11	IEEE802.11a	IEEE802.11b	IEEE802.11g	IEEE802.11n
Speed (Mbps)	2	54	11	54	300
Frequency band (GHz)	2.4	5	2.4	2.4	2.4
Finalized in	1997	1999	1999	2003	Not finalized yet
Indoor coverage range (m)		~15	~45	~45	~100
Outdoor coverage range (m)		~30	~90	~90	~400
Advantages	• First standard	• Avoid interference with other devices such as cordless phones and microwave oven	• Less expensive • Most popular before 2003 • Has been around the longest • Longer coverage range than 11a	• Compatible with IEEE802.11b devices which are still widely used • Faster than IEEE802.11b	• Compatible with IEEE802.11b and IEEE802.11g • Faster than IEEE b and g

REFERENCES

[1] K. Poynter, Review of Information Security at HM Revenue and Customs: Final Report, 2008. http://webarchive.nationalarchives.gov.uk/+/http://www.hmtreasury.gov.uk/media/0/1/poynter_review250608.pdf.

[2] B. Bahli, Y. Benslimane, An exploration of wireless computing risks, Inform. Manage. Comput. Security 12 (3) (2004) 245–254.

[3] J. Edney, W. Arbaugh, Real 802.11 Security, Addison-Wesley, Boston, MA, 2004.

[4] T. Swaminatha, C. Elden, Wireless Security and Privacy, Addison-Wesley, Boston, MA, 2003.

[5] Professional Information Security Association and Hong Kong Wireless Technology Industry Association, Hong Kong and Macau Wi-Fi Security Survey (War Driving) 2007, 2008.

[6] Professional Information Security Association and Hong Kong Wireless Technology Industry Association, Hong Kong and Macau Wi-Fi Security Survey (War Driving) 2008, 2009.

[7] G. Valadon, F. Le Goff, C. Berger, Daily Walks in Paris: A Practical Analysis of Wi-Fi Access Points, ACM, New York, NY, 2007.

[8] Viruslist.com, 2009. http://www.viruslist.com/en/find?words=war+driving+.

[9] ZerOne Security Team, War-Driving 2008. 2008. http://bigpack.blogbus.com/files/12265721420.pdf.

[10] H. Berghel, Wireless infidelity I: war driving, Commun. ACM 47 (9) (2004) 21–26.

[11] Reuters News, Wireless Networks Easy to Hack. 8 August 2005. http://www.itweb.co.za/sections/internet/2005/0508081002.asp?S=Reuters&A=REU&O=FPW.

[12] Associate Press, Florida Man Charged with Stealing Wi-Fi: Practice Is Common, but Arrests Are Unusual. 6 July 2005. http://www.msnbc.msn.com/id/8489534.

[13] A. Leary, Wi-Fi cloaks a new breed of intruder, St. Petersburg Times 2005. July 4.

[14] J. Wakefield, Wireless Hijacking under Scrutiny, 2005. http://news.bbc.co.uk/1/hi/technology/4721723.stm.

[15] M. Maxim, D. Pllio, Wireless Security, McGraw-Hill, New York, NY, 2002.

[16] A. Orebaugh, Wireshark & Ethereal Network Protocol Analyzer Toolkit, Syngress, Rockland, MA, 2007.

[17] C. Sanders, Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems, No Starch Press, San Francisco, CA, 2007.

[18] B. Ives, K.R. Walsh, H. Schneider, The domino effect of password reuse, Commun. ACM 47 (4) (2004) 75–78.

[19] J. Chen, M. Jiang, Y. Liu, Wireless LAN security and IEEE802.11i, IEEE Wireless Commun. 12 (1) (2005) 27–36.

[20] J. McCullough, Caution! Wireless Networking, Wiley, Chichester, 2004.

[21] H.I. Bulbul, I. Batmaz, M. Ozel, Wireless Network Security: Comparison of WEP (Wired Equivalent Privacy) Mechanism, WPA (Wi-Fi Protected Access) and RSN (Robust Security Network) Security Protocols, e-Forensics, Scotland, 2008.

[22] R. McMillan, ‘‘BlueBag’’ PC Sniffs out Bluetooth Flaws, IDG News Services, San Francisco, CA, 2006.

[23] M. Repo, Going Around with Bluetooth in Full Safety, F-Secure, Helsinki, 2006.

[24] Computer Weekly, The Cybermen Fight Back. 2006. http://www.bcs.org/server.php?show=conWebDoc.5959.

[25] M. Bialoalowv, Bluetooth Security Review, 2005. http://www.securityfocus.com/infocus/1830.

[26] B. Hopkins, R. Antony, Bluetooth for Java, Apress, New York, NY, 2003.

[27] Lakshmi, Bluetooth Standard and IPv6, 2008. http://www.ipv6.com/articles/applications/Bluetooth.htm.
[28] Y. Shaked, A. Wool, Cracking the Bluetooth pin, in: Proceedings of 3rd USENIX/ACM Conference on Mobile Systems, Applications, and Services, 2005.
[29] M. Herfurt, Detecting and Attacking Bluetooth-Enabled Cellphones at the Hannover Fairground, CeBIT, Hannover, 2004.
[30] B. Laurie, A. Laurie, Serious Flaws in Bluetooth Security Lead to Disclosure of Personal Data, A.L. Digital Ltd, London, 2004. http://www.thebunker.net/security/bluetooth.htm.
[31] K. Zetter, Security Cavities Ail Bluetooth, http://www.wired.com/politics/security/news/2004/08/64463.
[32] G. Cova, X. Huagang, G. Qiang, E. Guerrero, R. Ricardo, J. Estevez, A perspective of state-of-the-art wireless technologies for e-health applications,IT in Medicine & Education 1 (2009) 76–81.
[33] B. Schneier, CardSystems Exposes 40 Million Identities. Schneier on Security, June 2005. http://www.schneier.com/blog/archives/2005/06/cardsystems_exp.html.
[34] D. Tynan, Computer Privacy, O'Reilly, Sebastopol, CA, 2005.
[35] H. Berghel, Phishing mongers and posers, Commun. ACM 49 (4) (2006) 21–25.
[36] D. Llett, Mafia Insiders Infiltrating Firms, 2006. http://News.Com.
[37] C. Hurley, R. Rogers, F. Thornton, B. Baker, WarDriving and Wireless Penetration Testing, Syngress, Boston, MA, 2007.
[38] J.C. Sipior, B.T. Ward, Unintended invitation: organizational Wi-Fi use by external roaming users, Commun. ACM 50 (8) (2007) 72–77.
[39] IBM Corporation, Security Research: Wireless Security Auditor (WSA), 2009.
[40] E. Cayirci, Security in wireless ad hoc and sensor networks, Wiley, Chichester, 2009.
[41] G. Schudel, Router Security Strategies: Securing IP Network Traffic Planes, 2008.
[42] J.R. Vacca, Guide to Wireless Network Security, Springer, New York, NY, 2006.

Brain–Computer Interfaces for the Operation of Robotic and Prosthetic Devices

DENNIS J. McFARLAND

Laboratory of Neural Injury and Repair, Wadsworth Center, New York State Department of Health, Albany, New York, USA

JONATHAN R. WOLPAW

Laboratory of Neural Injury and Repair, Wadsworth Center, New York State Department of Health, Albany, New York, USA

Abstract

A brain–computer interface (BCI) uses signals recorded from the brain to convey the user's intent. BCIs can be used for communication or can provide control signals for robotic and prosthetic devices. In studies to date, both invasive and noninvasive recording methods have proved effective and have reached comparable levels of performance. The major challenge for both invasive and noninvasive BCI-based robotic control is to achieve the speed, accuracy, and reliability necessary for real-world applications. These requirements vary with the specific application and with the control strategy employed.

1. Introduction 170
2. Brain–Computer Interface Research and Development 171
3. BCI Movement Control 176
4. BCI Operation of Robotic and Prosthetic Devices 178
5. Current and Future Developments in BCI Movement Control 180

ADVANCES IN COMPUTERS, VOL. 79
ISSN: 0065-2458/DOI: 10.1016/S0065-2458(10)79004-5

6. Conclusion 183
Acknowledgments 184
References 184

1. Introduction

The electrical activity of the brain produces signals that are detectable on the scalp, on the cortical surface, or within the brain. Brain–computer interfaces (BCIs) translate these signals into outputs that communicate a user's intent without the participation of peripheral nerves and muscles [1] (see Fig. 1). Because they do not depend on neuromuscular control, BCIs can provide communication and control options for people with devastating neuromuscular disorders such as amyotrophic

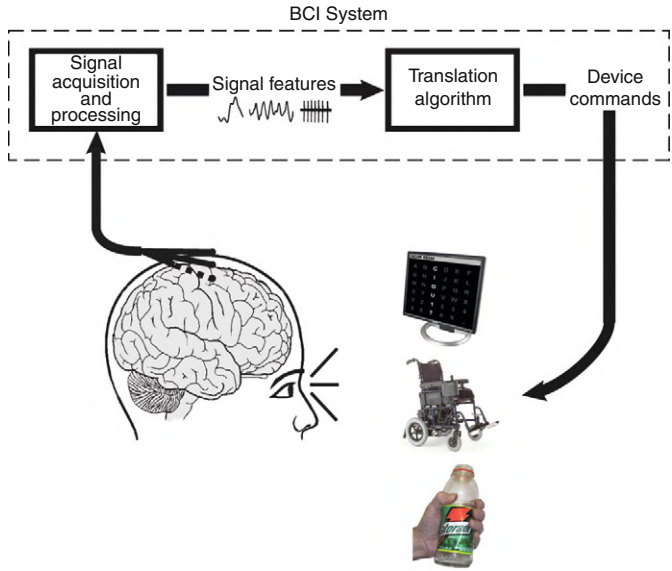

FIG. 1. Basic design and operation of any BCI system. Signals from the brain are acquired by electrodes on the scalp or brain and processed to extract signal features that reflect the user's intent. These features are translated into commands to operate a device. The user must develop and maintain good correlation between his/her intent and the signal features employed by the BCI; and the BCI must select and extract features that the user can control and must translate those features into device commands (adapted from Ref. [1]).

lateral sclerosis (ALS), brainstem stroke, cerebral palsy, and spinal cord injury. The central purpose of BCI research and development is to enable these users to convey their wishes to caregivers, to use word-processing programs and other software, or even to control a robotic arm or a neuroprosthesis. More recently, there has been speculation that BCIs could be useful to people with lesser, or even no, motor impairments.

It has been known since the pioneering work of Hans Berger over 80 years ago that the electrical activity of the brain can be recorded noninvasively by electrodes on the surface of the scalp [2]. Berger observed that a rhythm of about 10 Hz, reactive to light, was prominent on the posterior scalp. He called it the alpha rhythm. This and other observations by Berger showed that the electroencephalogram (EEG) could serve as an index of the gross state of the brain.

Electrodes on the surface of the scalp are at a distance from the source of the signals in brain tissue and are separated from it by the coverings of the brain, the skull, subcutaneous tissue, and the scalp. As a result, these signals are considerably degraded and only the synchronized activity of large numbers of neural elements can be detected. This limits the resolution with which brain activity can be monitored. Furthermore, scalp electrodes also pick up activity from sources other than the brain. These other sources include environmental noise (e.g., 50- or 60-Hz activity from power lines) and biological noise (e.g., electrical activity from the heart, skeletal muscles, and eyes). Nevertheless, since the time of Berger, many studies have used the EEG very effectively to gain insight into brain function. Many of these studies have used averaging to separate EEG from superimposed electrical noise.

There are two major paradigms used in EEG research. Evoked potentials are transient waveforms that are phase-locked to an event such as a visual stimulus. They are typically analyzed by averaging many similar events in the time domain. The EEG also contains oscillatory features. Although these oscillatory features may occur in response to specific events, they are usually not phase-locked and are typically studied by spectral analysis. Historically, most EEG studies have examined phase-locked evoked potentials. Both of these major paradigms have been studied in BCI research (see Ref. [1] for review).

2. Brain–Computer Interface Research and Development

The term *brain–computer interface* originated with Vidal [3] who devised a BCI system based on visual evoked potentials [3]. His users viewed a diamond-shaped red checkerboard illuminated with a xenon flash. By attending to different corners of

the flashing checkerboard, Vidal's users could generate right, up, left, or down commands. This enabled them to move through a maze presented on a graphic terminal. The EEG was digitized and analyzed by an IBM360 mainframe and the experimental events were controlled by a XDS Sigma 7 computer. The users first provided data to train a stepwise linear discriminant function. The discriminant function was then applied online in real-time to allow navigation of the maze. Thus, Vidal [3] used signal-processing techniques to realize real-time analysis of the EEG with minimal averaging. The waveforms shown by Vidal [3] suggest that his BCI used EEG activity in the time frame of the N100–P200 (i.e., negative and positive peaks at approximately 100 and 200 ms, respectively) components [4].

Vidal's achievement was an interesting demonstration or proof of principal. At the time, it was far from practical given that it depended on a shared mainframe system with limited processing capacity. Vidal [3] also included in his system online removal of non-EEG (i.e., ocular) artifacts. Somewhat earlier, Dewan [5] instructed users to explicitly use eye movements in order to modulate their brain waves. He showed that subjects could learn to transmit Morse code messages using EEG activity associated with eye movements. The fact that both the Dewan and Vidal BCIs depended on eye movements made them somewhat less interesting from scientific or clinical points of view, since they required actual muscle control (i.e., eye movements) and simply used EEG to reflect the resulting gaze direction.

Farwell and Donchin [6] reported the first use of an EEG-based spelling device that used the P300 evoked potential (see Fig. 2A). Their users viewed a 6×6 matrix of the letters of the alphabet and several other symbols. They focused attention on the desired selection as the rows and columns of the matrix were repeatedly flashed to elicit evoked potentials. Whenever the desired selection flashed, it elicited a P300 evoked potential from the brain (i.e., a positive voltage peak at about 300 ms) [4]. By detecting this response, the BCI system was able to recognize which item the user wanted to select. Farwell and Donchin [6] found that their users were able to spell the word ''brain'' with the P300 spelling device. In addition, they did an offline comparison of detection algorithms and found that the stepwise linear discriminant analysis was generally best. The fact that the P300 potential reflects attention rather than simply gaze direction implied that this BCI did not depend on muscle (i.e., eye-movement) control. Thus, it represented a significant advance. Several groups have further developed this BCI method (e.g., [7–9]).

Wolpaw et al. [10] reported the first use of sensorimotor rhythms (SMRs) (i.e., 9–13 Hz activity recorded from central scalp locations associated with motor function) for cursor control (see Fig. 2B). SMRs are EEG rhythms that are modulated by movement or the imagination of movement and are spontaneous in the sense that they do not require specific stimuli for their occurrence. People learned to use SMRs to move a cursor to hit a target located on the top or bottom edge of a video screen.

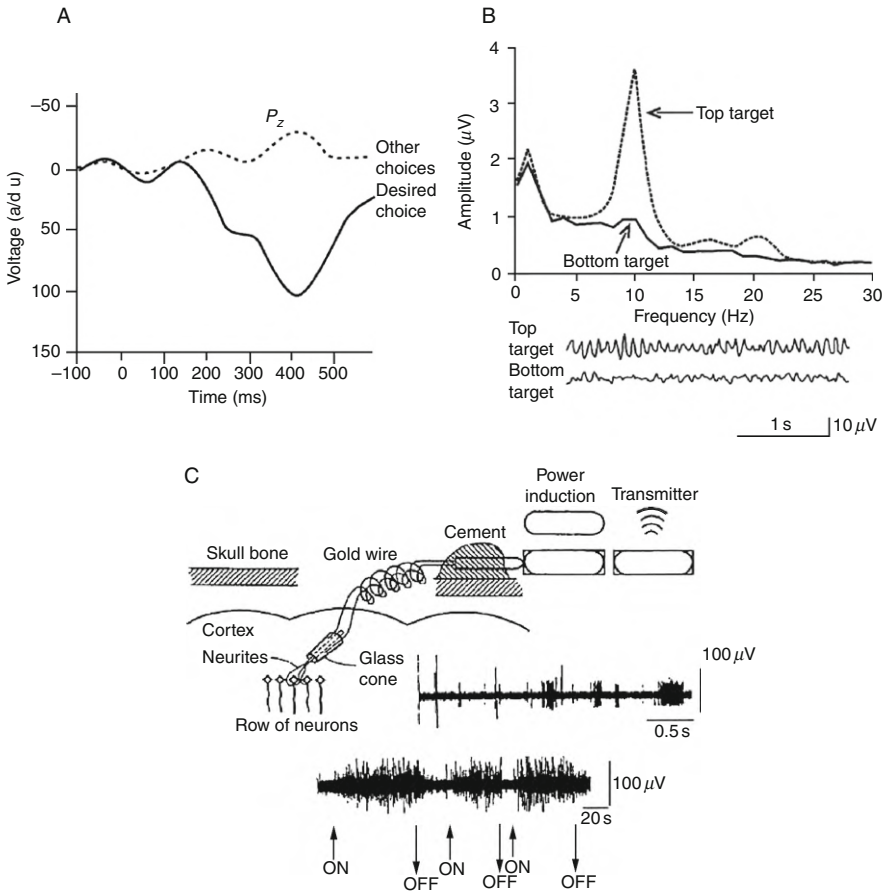

FIG. 2. Present-day human BCI systems. (A, B) Noninvasive and (C) invasive. (A) P300 BCI. A matrix of possible selections is presented on a screen and scalp EEG is recorded while groups of these selections flash in succession. The BCI system determines which choice evokes a P300 response (see text). (B) Sensorimotor rhythm (SMR) BCI. Scalp EEG is recorded over sensorimotor cortex. Users control SMR amplitudes to move a cursor to a target on the screen (see text). (C) Cortical neuronal BCI. Electrodes implanted in cortex detect action potentials of single neurons. Users control neuronal firing rates to move a cursor to a target on the screen (see text) (adapted from Ref. [1]).

Cursor movement was controlled by SMR amplitude (measured by spectral analysis). A distinctive feature of this task is that it required the user to rapidly switch between two states in order to select a particular target. The rapid bidirectional nature of the Wolpaw et al. [10] paradigm made it distinct from prior studies that

produced long-term unidirectional changes in brain rhythms as a new therapeutic technique (e.g., [11]). A series of subsequent studies showed that the signals that controlled the cursor were actual EEG activity and also that covert muscle activity did not mediate this EEG control (e.g., [12, 13]).

These initial SMR results were subsequently replicated by others (e.g., [14, 15]) and extended to multidimensional control [16, 17]. Together, these P300 and SMR BCI studies showed that noninvasive EEG recording of brain signals can serve as the basis for communication and can control devices.

More recently, a number of laboratories have explored the possibility of developing BCIs using single-neuron activity detected by microelectrodes implanted in cortex (e.g., [18, 19]; see Fig. 2C). Much of this research had been done in nonhuman primates. However, there have been trials in human users (e.g., [18]). Other studies have shown that recordings of electrocorticographic (ECoG) activity from the surface of the brain can also provide signals for a BCI (e.g., [20]). To date these studies have indicated that invasive recording methods could also serve as the basis for BCIs. At the same time, important issues concerning their suitability for long-term human use remain to be resolved.

Communication and control applications are interactive processes that require the user to observe the results in order to maintain good performance and to correct mistakes. For this reason, actual BCIs need to run in real-time and to provide real-time feedback to the user. While many early BCI studies satisfied this requirement (e.g., [10, 15]), later studies have often been based on offline analyses of prerecorded data (e.g., [21]). For example, in the Lotte et al. [22] review of studies evaluating BCI signal classification algorithms, most had used offline analyses. Indeed, the current popularity of BCI research is probably due in part to the ease with which offline analyses can be performed on publicly available data sets. While such offline studies may help to guide actual online BCI investigations, there is no guarantee that offline results will generalize to online performance. The user's brain signals are often affected by the BCI's outputs, which are in turn determined by the algorithm that the BCI is using. Thus, it is not possible to predict results exactly from offline analyses, which cannot take those effects into account.

Muller and Blankertz [14] have advocated a machine-learning approach to BCI in which a statistical analysis of a calibration measurement is used to train the system. The goal of this approach is to develop a ''zero-training'' method that provides effective performance from the first session. They contrast this approach with one based on training users to control specific features of brain signals (e.g., [10]). A system that can be used without extensive training is appealing since it requires less initial effort on the part of both the BCI user and the system operator. The operation of such a system is based upon the as yet uncertain premise that the user can repeatedly and reliably maintain the specified correlations between brain signals

and intent. Figure 3 presents three different conceptualizations of where adaptation might take place to establish and maintain good BCI performance. In the first the BCI adapts to the user, in the second the user adapts to the BCI, and in the third both user and system adapt to each other.

It is certainly possible to devise a system that does not require training on the part of the user. From a practical standpoint, the issues are whether such a system is optimal and whether any benefit from extensive training is worth the time that is required for the user. Certainly, it is desirable for a BCI system to operate as rapidly and accurately as possible when first used by the user. However, adaptation on the part of the user is inevitable. If continued adaptation by the system is part of normal use then it has no real cost for the user. To the extent that system adaptation requires knowledge of the users' intent, it may be necessary to periodically conduct some kind of calibration run. In this case, there is a cost in terms of user time and effort. However, disabled individuals who would actually benefit from a BCI system are not casual users and would presumably be willing to invest time and effort to optimize operation.

A number of BCI systems are designed to detect the user's performance of specific cognitive tasks. Curran et al. [23] suggest that cognitive tasks such as navigation and auditory imagery might be more useful in driving a BCI than motor imagery. However, SMR-based BCIs may have several advantages over systems that depend on complex cognitive operations. For example, the structures involved in auditory imagery (i.e., the subjective experience of hearing in the absence of auditory stimulation) are also likely to be driven by auditory sensory input.

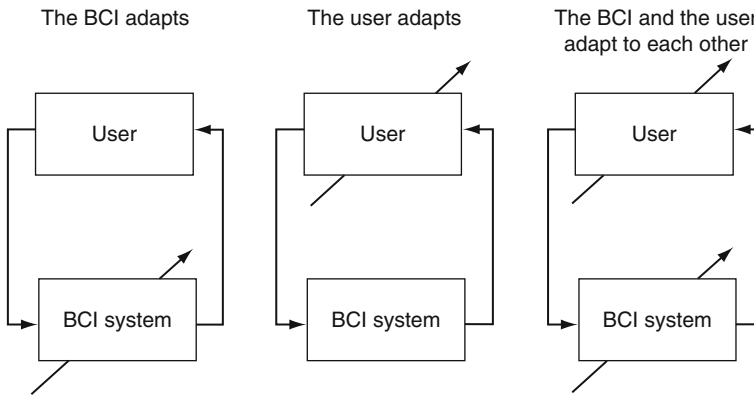

FIG. 3. Three approaches to BCI design. The arrows indicate whether the BCI, the user, or both adapt to optimize and maintain BCI performance (adapted from Ref. [12]).

Wolpaw and McFarland [17] report that with extended practice users report that motor imagery is no longer necessary to operate an SMR-based BCI. As is typical of many simple motor tasks, performance becomes automatized with extended practice. Automatized performance may be less likely to interfere with mental operations that users might wish to engage in concurrent with BCI use. For example, composing a manuscript is much easier if the writer does not need to think extensively about each individual keystroke. Thus a BCI might be more practical if it does not depend upon complex cognitive operations.

As noted above, EEG recording may be contaminated by nonbrain activity (e.g., line noise, muscle activity, etc.) [24, 25]. Activity recorded from the scalp represents the superposition of many signals, some of which originate in the brain and some of which originate elsewhere. These include potentials generated by the retinal dipoles (i.e., eye movements and eye blinks) and the facial muscles. It is noteworthy that mental effort is often associated with changes in eye blink rate and muscle activity (e.g., [26]). BCI users might generate these artifacts without being aware of what they are doing simply by making facial expressions associated with effort.

Facial muscles can generate signals with energy in the spectral bands used as features in an SMR-based BCI. Muscle activity can also modulate SMR activity. For example, a user could move his/her right hand to reduce SMR activity over the left hemisphere. This sort of mediation of the EEG by peripheral muscle movements was a concern during the early days of BCI development. As noted earlier, Dewan [5] trained users to send Morse code messages using occipital alpha rhythms (i.e., 9–12 Hz activity recorded from the back of the scalp and associated with visual functions) modulated by voluntary movements of eye muscles. For this reason, Vaughan et al. [13] recorded EMG from 10 distal limb muscles while BCI users used SMRs to move a cursor to targets on a video screen. EMG activity was very low in these well-trained users. Most importantly, the correlations between target position and EEG activity could not be accounted for by EMG activity. Similar studies have been done with BCI users moving a cursor in two dimensions [17]. These studies show that SMR modulation does not require actual movements or muscle activity.

3. BCI Movement Control

Movement control applications can be based on either kinematic control or goal selection. For kinematic control, the BCI specifies the exact movements of the device continuously and in real time. For goal selection, the BCI simply indicates the desired outcome, and downstream hardware and software handle the

continuous kinematic control that achieves the outcome. Goal selection is also known as inverse kinematic control since the specific control parameters are computed from knowledge of the goal. Goal selection is much less demanding in terms of the complexity and rate of the control signals the BCI needs to provide. For example, to control a robotic arm in three-dimensional space, kinematic control requires that the BCI provide three control signals continuously in real time, while goal selection simply requires that the BCI specify the final location of the hand part of the robotic arm. Goal selection might also convey more complex commands, such as ''make coffee.'' This would of course require a downstream device with detailed and continually updated knowledge of the environment. The advantage of goal-based control is that the BCI itself needs to provide much less information. On the other hand, kinematic control can provide more flexible control in a wider variety of circumstances. It is also possible to combine the two approaches (e.g., [27]).

The control requirements of robotic and prosthetic medical devices vary widely. A wheelchair could be operated with two independent control signals, for example, one for angular direction and another for speed of movement. At the same time, extremely high accuracy and reliability is essential if the wheelchair is to be used in a potentially hazardous environment (e.g., city traffic). In contrast, a prosthetic arm would need many more degrees of freedom if all of the joints are to be controlled, but accuracy and reliability may be less critical (e.g., picking up objects).

As discussed earlier, BCIs can be either noninvasive or invasive. Present-day noninvasive BCIs derive the user's intent from scalp-recorded electroencephalographic (EEG) activity. They are clearly able to provide simple communication and control to people with severe disabilities (e.g., [28–30]). Present-day invasive BCIs determine the user's intent from neuronal action potentials or local field potentials recorded from within the cerebral cortex or from its surface. They have been studied mainly in nonhuman primates and to a limited extent in people [18–20, 31–37]. These invasive BCIs face significant technical difficulties and entail clinical risks. Recording electrodes must be implanted in or on the cortex and must function well for long periods (i.e., many years), and they risk infection and other kinds of damage to the brain. The impetus to develop invasive BCI methods is based in part on the common belief [18, 38–42] that only invasive BCIs will be capable of providing their users with real-time multidimensional sequential control of a robotic arm or a neuroprosthesis.

Nevertheless, in an early study Wolpaw and McFarland [16] showed that a noninvasive BCI that uses scalp-recorded EEG activity (i.e., SMRs) can provide humans with multidimensional movement control. Furthermore, more recent studies [17, 43] showed that a noninvasive EEG-based BCI that incorporates an adaptive algorithm and other technical improvements can give humans two-dimensional movement control and sequential control comparable in movement time, precision,

and accuracy to that achieved by invasive BCIs in monkeys [32, 36] or humans [18]. Two-dimensional control with SMRs has been achieved by other groups as well (e.g., [44, 45]). Most recently, McFarland et al. [46] showed that an EEG-based BCI can provide three-dimensional movement control comparable to that achieved by invasive methods [19].

4. BCI Operation of Robotic and Prosthetic Devices

Chapin et al. [33] provided the first demonstration of direct robotic control using brain signals. They first trained rats to obtain water rewards while action potentials were recorded from neurons in motor cortex. Subsequently, four of six animals learned to use these brain signals alone (without actually pressing the bar) to move a robotic arm in one dimension to obtain water.

Carmena et al. [32] trained monkeys to operate a lever to position a cursor over a target and then adjust the cursor size so as to obtain juice rewards. During this initial phase, signals were recorded from neurons in motor cortex. Subsequently, the animals were able to use these brain signals to perform the task. Carmena et al. [32] report that the monkeys were able to perform the task even when their arms did not move. In a subsequent analysis of the Carmena et al. [32] data, Lebedev et al. [47] reported that as training progressed; the neuronal activity became less representative of actual hand movements and more representative of the robotic actuator. This suggests that the neuronal activity became tuned to the robotic task as a result of brain plasticity (i.e., adaptive changes in brain function such as learning). This group also developed a system combining neural signals with sensor-driven reflex-like reactions [48]. An offline simulation using the original monkey data showed that the shared control paradigm improved performance over that produced by neural signals alone.

Taylor et al. [49] compared neutrally controlled cursor and robotic arm movements in two monkeys trained to make reaching movements in virtual three-dimensional space while neuronal activity was recorded from motor cortex. Subsequently, the animals learned to control the cursor and robotic arm with neural activity alone. They found that robotic arm movement was slower than cursor movement and that the animals tended to make sharper turns with cursor movement (due to the mechanical properties of the robotic arm). This same group subsequently trained a different pair of monkeys to move a robotic arm in three dimensions, and in addition, to control a gripper [50]. This allowed the monkeys to perform a self-feeding task. The authors refer to this as ‘‘embodied prosthetic control.’’

To date, invasive studies in animals have shown that direct neural control of prosthetic devices is possible. These studies have been performed in animals with intact motor systems, and have relied on pretraining with actual movement. Recently, Hochberg et al. [18] trained a human user with a cortical implant to open and close a prosthetic hand and to control a simple robotic arm in two dimensions. This individual had a high spinal cord injury that precluded training the BCI with signals produced during actual hand movements. Neurons in motor cortex served as input to linear decoders that controlled the prosthesis or robotic arm. The authors state that their user was able to control the robotic arm within about 10 min and could use it to grip a piece of candy and deliver it to the hand of a technician.

Millan et al. [51] used noninvasive EEG signals recorded over sensorimotor areas to provide two human users with control of a mobile robot. These users imagined three mental states corresponding to ''turn left,'' ''turn right,'' and ''move forward.'' User commands were determined by a classifier that used spectral features from the EEG as input and were combined with robot sensor information to allow navigation. Both users successfully moved the robot through several rooms.

Galan et al. [52] describe a similar system that was used to control a simulation of the movement of a wheelchair through a virtual environment. This system incorporated intelligent control of the virtual wheelchair so that it only allowed for movement that was consistent with environmental information from the wheelchair's sensors. Galan et al. [52] state that their machine-learning approach allows for autonomous operation over long periods of time without adaptive algorithms. Excluding the first session, one of their user's performances varied between 50% and 100% correct choices during sessions of 10 trials while the other user varied between 40% and 80% correct choices. Thus, while the system did not require continuous adaptation, performance was quite variable.

At the Wadsworth Center we have extended our studies using SMRs for multidimensional and mouse-like control of cursor movements [17, 43, 46] to control of a simple robotic arm [53]. In prior training, the user learned to move the cursor in two dimensions from the center of the screen to a target at one of 20 possible locations on the periphery of the screen. Then, the robotic arm was placed in front of the screen and its movements were substituted for those of the cursor. Performance with the robotic arm was comparable to that with the cursor both in the percent of trials completed within the allotted time and in the median time per trial. The median time was slightly longer for the arm because its movement was slowed to accommodate its tendency for mechanical oscillations.

Bell et al. [54] described a humanoid robot controlled by the P300 response of a human operator. The robot was equipped with a video camera that displayed objects in its environment to the human user. These objects were sequentially and repeatedly

outlined in the user's display and the EEG response was used to determine which object the user wished to select. Based on this information, the robot would pick up the desired object. Accuracy improved with more repetitions. The authors report that the robot could select from among four possible choices in 5 s with 95% accuracy.

Pfurtscheller et al. [55] trained a tetraplegic patient to control a hand orthosis (i.e., a device applied to a human limb to control or enhance movement) with EEG recorded over sensorimotor cortex. The patient was originally trained to use motor imagery to produce changes in the EEG. Following this initial training, the patient could use these EEG signals to control the opening and closing of his normally paralyzed hand by the orthosis. Muller-Putz et al. [29] trained a different patient with a spinal cord lesion to control an implanted neuroprosthesis. Again, the patient was originally trained to use motor imagery to modulate EEG. The patient then used his EEG to step through several phases of a hand grip.

Collaboration between RIKEN and TOYOTA produced a BCI system based upon motor imagery that could control a wheelchair [56]. These investigators used left-hand and right-hand imagery for lateral control and foot imagery for forward control. In addition, they added EMG from a cheek muscle to provide a reliable stop signal.

In summary, BCI studies using both invasive and noninvasive recording of brain activity have been successful in demonstrating the feasibility of controlling robotic devices. However, for the most part, these studies have only been demonstrations of the potential use of BCI technology for robotic and prosthetic applications. The practical application of invasive approaches will require solutions to problems of long-term recording stability and safety. Noninvasive methods require development of recording methodology that can meet the needs of everyday use and can be supported by caregivers who are not skilled technicians. Both methodologies will need to improve their speed, accuracy, and reliability in order to provide truly useful robotic or prosthetic control.

5. Current and Future Developments in BCI Movement Control

To date, most groups using invasive BC methods to control movements have used direct kinematic control. This is probably due to the basic-science research tradition from which these researchers come: they have traditionally focused on understanding the neural basis of movement control by studying neuronal correlates of kinematic parameters. Noninvasive studies have been more mixed; while some have developed kinematic control, others have pursued a goal-selection approach.

Robotic and prosthetic applications do not appear to represent a unique challenge for BCI technology. Investigators using both invasive methods and noninvasive methods achieved a smooth transition from paradigms that control cursor movements to those that control actual mechanical devices. The major problem for BCI applications is in providing fast, accurate, and reliable control signals. Certainly, developments in robotics will be useful for systems used by actual patients, but these developments will not by themselves solve the problems that are specific to BCI development.

The practical use of kinematic control may require a larger number of independent control signals than are currently available from BCI technology. For example, robotic arms often have seven or more degrees of freedom, and the human hand and arm have even more. As noted above, invasive methods have achieved three dimensions of movement control plus grasp control [50] and noninvasive methods have achieved two dimensions of movement control plus selection (i.e., grasp) control [43] and three dimensions of movement control [46]. At the same time, the goal-selection strategy (i.e., reverse kinematic control) is certainly applicable to robotic applications with current BCI methodology, and may provide a good alternative for control of complex and sequential movements.

Several commercial endeavors have recently produced inexpensive devices that are purported to measure EEG. Both Emotiv and Neurosky have developed products with a limited number of electrodes that do not use conventional gel-based recording technology [57]. These devices are intended to provide input for video games. It is not clear to what extent these devices use actual EEG activity as opposed to scalp muscle activity or other nonbrain signals. Given the well-established prominence of EMG activity in activity recorded from the head, it seems likely that such signals account for much of the control produced by these devices.

Both noninvasive and invasive methods could benefit from improvements in recording methods. Current invasive methods have not yet dealt adequately with the need for long-term performance stability. The brain's complex reaction to an implant is still imperfectly understood and can impair long-term performance. Current noninvasive (i.e., EEG) electrodes require a certain level of skill in the person placing them, and require periodic maintenance to maintain sufficiently good contact with the skin. More convenient and stable electrodes are under development. Improved methods for extracting key EEG features, for translating those features into device control, and for training users should also help to improve BCI performance.

Developments in robotic technology may also facilitate BCI use. Kim et al. [58] identify several potential areas of interest. These include shared control between the user and robotic device, impedance control in the robotic device, and soft actuators. As discussed earlier, shared control involves combining user commands with device

intelligence (e.g., [48,52]). Kim et al. [58] note that the neuromuscular system adaptively modulates mechanical impedance in a manner appropriate for the task at hand. BCI signals could be used to control this aspect of robotic arm operation as well as positioning. Soft actuators deal with the safety issues inherent in human interactions while at the same time maintaining adequate performance.

Recent developments in computer hardware have provided compact portable systems that are extremely powerful. Use of digital electronics has also lead to reductions in the size and improvements in the performance of EEG amplifiers. Thus, it is no longer necessary to use a large time-shared mainframe as was the case for Vidal [3]. Standard laptops can easily handle the vast majority of real-time BCI protocols. There have also been improvements in signal-processing and machine-learning algorithms. Coupled with the discovery of new EEG features for BCI use and the development of new paradigms for user training, these improvements are gradually increasing the speed and reliability of BCI communication and control. These developments can be facilitated by the use of a standard BCI-related software platform such as BCI2000 [59].

BCI2000 is a general-purpose research and development system that can readily incorporate alone or in combination any brain signals, signal-processing methods, output devices, and operating protocols. BCI2000 consists of a general standard for creating interchangeable modules designed according to object-oriented principles (see Fig. 4). These consist of a source module for signal acquisition, a signal-processing module, and a user application. Configuration and coordination of

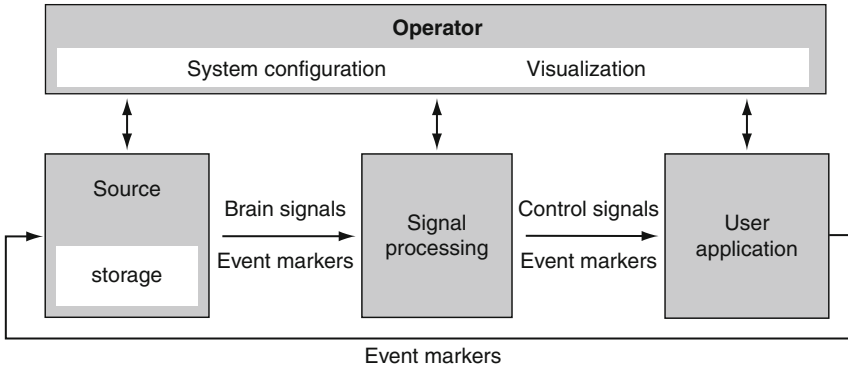

FIG. 4. BCI2000 design. BCI2000 consists of four modules: operator, source, signal processing, and application. The operator module handles system configuration and online presentation of results to the operator or investigator. During operation, information is communicated from source to signal processing to user application and back to source (adapted from Ref. [59]).

these three modules is accomplished by a fourth operator module. To date many different source modules, signal-processing modules, and user applications have been created for the BCI2000 standard (see http://www.bci2000.org/BCI2000/Home.html).

A number of specific realizations of the BCI2000 standard are available and illustrate the utility of this approach. EEG is recorded with a number of different commercial amplifiers. In some cases, the computer is interfaced with the EEG amplifiers through analogue-to-digital (A/D) converters such as the Data Translation DT3000 series PCI boards. In other cases, such as the g.USBamp by g.tec, the amplifier and A/D converter are a single unit that interfaces via USB ports. Acquisition modules are available for both of these cases, as well as for several others [60]. These acquisition modules work interchangeably with a number of different signal-processing modules. Oscillatory signals can be quantified by Fourier-based spectral analysis, autoregressive spectral analysis, or digital filtering. Interchangeable signal-processing modules are available for each of these methods. Finally, there are several distinct realizations of the BCI2000 application module, as the specific task performed by the BCI user may vary. Since the source code is readily available online (http://www.bci2000.org/BCI2000/Home.html), these modules can be modified. Alternatively, the documentation describes how to create entirely new modules. In the latter case, the advantage of the BCI2000 standard is that only the new module needs to be created since the others will still be completely functional and will interface with it seamlessly.

The speed and accuracy of current BCI technology limits its utility to users who have few other options. As such, developments in the next 10 years will probably focus on this group of users and involve technical issues related to making systems suitable for home use. This will require BCI systems that are robust and usable by nonexpert caregivers. Use of BCI devices by individuals without severe motor impairments will depend upon new scientific developments that are currently unpredictable. Devices that claim to use EEG but probably rely on artifacts (e.g., electrical signals from muscles or the eyes) will probably continue to appear on the market.

6. Conclusion

How far BCI technology will develop and how useful it will become depend on the success and focus of future research and development. At present, it is apparent that BCIs can serve the basic communication needs of people whose severe motor disabilities prevent them from using conventional augmentative communication devices.

Current noninvasive (i.e., EEG) methods require a certain level of skill in the person placing them, and require periodic maintenance to maintain sufficiently good contact with the skin. More convenient and stable electrodes are under development. Improved methods for extracting key EEG features, for translating those features into device control, and for training users should also help to improve BCI performance. Current invasive methods have not yet resolved all the issues associated with long-term use. Research now underway is attempting to do so. Invasive and noninvasive BCIs have achieved similar levels of multidimensional movement control and both approaches warrant continued research efforts.

Up to the present, almost all BCI research has taken place in the laboratory. If BCIs are to become important new communication and control options for people with motor disabilities, careful and comprehensive clinical research is essential. Furthermore, if widespread use is to become possible, the practical issues of economic feasibility and technical support must be satisfactorily addressed.

ACKNOWLEDGMENTS

Work in the authors' laboratory has been supported by grants from NIH (HD30146 (NCMRR/NICHD) and EB00856 (NIBIB & NINDS)), the James S. McDonnell Foundation, the Altran Foundation, the ALS Hope Foundation, and the Brain Communication Foundation. We thank Stefan Winden for his helpful comments on this chapter.

REFERENCES

[1] J.R. Wolpaw, N. Birbaumer, D.J. McFarland, G. Pfurtscheller, T.M. Vaughan, Brain–computer interfaces for communication and control, Clin. Neurophysiol. 113 (2002) 767–791.

[2] E. Neidermeyer, Historical aspects, in: E. Neidermeyer, F. Lopes da Silva (Eds.), Electroencephalography: Basic Principals, Clinical Applications, and Related Fields, fifth Ed., Lippincott Williams and Wilkins, Philadelphia, PA, 2005, pp. 1–15.

[3] J.J. Vidal, Real-time detection of brain events in EEG, Proc. IEEE 65 (1977) 633–641.

[4] C.S. Herrmann, R.T. Knight, Mechanisms of human attention: event-related potentials and oscillations, Neurosci. Biobehav. Rev. 25 (2001) 465–476.

[5] E.M. Dewan, Occipital alpha rhythm eye position and lens accommodation, Nature 214 (1967) 975–977.

[6] L.A. Farwell, E. Donchin, Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials, Electroencephalogr. Clin. Neurophysiol. 70 (1988) 510–523.

[7] J.D. Bayliss, S.A. Inverso, A. Tentler, Changing the P300 brain computer interface, Cyberpsychol. Behav. 7 (2004) 694–704.

[8] C. Guger, S. Daban, E. Sellers, C. Holzner, G. Kraus, R. Carabalona, F. Gramatica, G. Edlinger, How many people are able to control a P300-based brain–computer interface (BCI)? Neurosci. Lett. 462 (2009) 94–98.

[9] D.J. Krusienski, E.W. Sellers, D.J. McFarland, T.M. Vaughan, J.R. Wolpaw, Toward enhanced P300 speller performance, J. Neurosc. Methods 167 (2008) 15–21.

[10] J.R. Wolpaw, D.J. McFarland, G.W. Neat, C.A. Forneris, An EEG-based brain–computer interface for cursor control, Electroencephalogr. Clin. Neurophysiol. 78 (1991) 252–259.
[11] M.B. Sterman, L.R. MacDonald, R.K. Stone, Biofeedback training of sensorimotor EEG in man and its effect on epilepsy, Epilepsia 15 (1974) 395–416.
[12] D.J. McFarland, W.A. Sarnacki, T.M. Vaughan, J.R. Wolpaw, Brain–computer interface (BCI) operation: signal and noise during early training sessions, Clin. Neurophysiol. 116 (2005) 56–62.
[13] T.M. Vaughan, L.A. Miner, D.J. McFarland, J.R. Wolpaw, EEG-based communication: analysis of concurrent EMG activity, Electroencephalogr. Clin. Neurophysiol. 107 (1998) 428–433.
[14] K.-R. Muller, B. Blankertz, Towards noninvasive brain–computer interfaces, IEEE Signal Process. Mag. 23 (2006) 125–128.
[15] G. Pfurtscheller, D. Flotzinger, J. Kalcher, Brain–computer interface—a new communication device for handicapped persons, J. Microcomput. Appl. 16 (1993) 293–299.
[16] J.R. Wolpaw, D.J. McFarland, Multichannel EEG-based brain–computer communication, Electroencephalogr. Clin. Neurophysiol. 90 (1994) 444–449.
[17] J.R. Wolpaw, D.J. McFarland, Control of a two-dimensional movement signal by a non-invasive brain–computer interface, Proc. Natl. Acad. Sci. USA 51 (2004) 17849–17854.
[18] L.R. Hochberg, M.D. Serruya, G.M. Friehs, J.A. Mukand, M. Saleh, A.H. Caplan, A. Branner, D.R. D. Penn, J.P. Donoghue, Neuronal ensemble control of prosthetic devices by a human with tetraplegia, Nature 442 (2006) 164–171.
[19] D.A. Taylor, S. Helms Tillery, A.B. Schwartz, Direct cortical control of 3D neuroprosthetic devices, Science 296 (2002) 1829–1832.
[20] E.C. Leuthardt, G. Schalk, J.R. Wolpaw, J.G. Ojemann, D.W. Moran, A brain–computer interface using electrocorticographic signals in humans, J. Neural Eng. 1 (2004) 63–71.
[21] B. Blankertz, K.-R. Muller, D.J. Krusienski, G. Schalk, J.R. Wolpaw, A. Schlogl, G. Pfurtscheller, J. Millan, M. Schroder, N. Birbaumer, The BCI competition III: validating alternative approaches to actual BCI problems, IEEE Trans. Neural Syst. Rehabil. Eng. 14 (2006) 153–159.
[22] F. Lotte, M. Congedo, A. Lecuyer, F. Lamarche, B. Arnaldi, A review of classification algorithms for EEG-based brain–computer interfaces, J. Neural Eng. 4 (2007) 1–13.
[23] E. Curran, P. Sykacek, M. Stokes, S.J. Roberts, W. Penny, I. Johnsrude, A. Owen, Cognitive tasks for driving a brain–computer interface system: a pilot study, IEEE Trans. Neural Syst. Rehabil. Eng. 12 (2004) 48–54.
[24] M. Fatourechi, A. Bashashati, R.K. Ward, G.E. Birch, EMG and EOG artifacts in brain computer interface systems: a survey, Clin. Neurophysiol. 118 (2007) 480–494.
[25] I.I. Goncharova, D.J. McFarland, T.M. Vaughan, J.R. Wolpaw, EMG contamination of EEG: spectral and topographical characteristics, Clin. Neurophysiol. 114 (2003) 1580–1593.
[26] E.M. Whitham, T. Lewis, K.J. Pope, S.P. Fitzbibbon, C.R. Clark, S. Loveless, D. DeLosAngeles, A.K. Wallace, M. Broberg, J.O. Willoughby, Thinking activates EMG in scalp electrical recordings, Clin. Neurophysiol. 119 (2008) 1166–1175.
[27] R. Boulic, D. Thalmann, Combined direct and inverse kinematic control for articulated figure motion editing, Comput. Graph. 11 (1992) 189–202.
[28] N. Birbaumer, N. Ghanayim, T. Hinterberger, I. Iversen, B. Kotchoubey, A. Kubler, J. Perlmouter, E. Taub, H. Flor, A spelling device for the paralyzed, Nature 398 (1999) 297–298.
[29] G.R. Muller-Putz, R. Scherer, G. Pfurtscheller, R. Rupp, EEG-based neuroprosthesis control: a step towards clinical practice, Neurosci. Lett. 382 (2005) 169–174.
[30] T.M. Vaughan, D.J. McFarland, G. Schalk, W.A. Sarnacki, D.J. Krusienski, E.W. Sellers, J.R. Wolpaw, The Wadsworth BCI research and development program: at home with BCI, IEEE Trans. Neural Syst. Rehabil. Eng. 14 (2006) 229–233.

[31] R.A. Andersen, S. Musallam, B. Pesaran, Selecting signals for a brain–machine interface, Curr. Opin. Neurobiol. 14 (2004) 720–726.
[32] J.M. Carmena, M.A. Lebedev, R.E. Crist, J.E. O'Doherty, D.M. Santucci, D.F. Dimitrov, P.G. Patil, C.S. Henriquez, M.A.L. Nicolelis, Learning to control a brain–machine interface for reaching and grasping by primates, PLoS Biol. 1 (2003) 193–208.
[33] J.K. Chapin, K.A. Moxon, R.S. Markowitz, M.A.L. Nicolelis, Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex, Nat. Neurosci. 2 (1999) 664–670.
[34] P.R. Kennedy, R.A.E. Bakay, M.M. Moore, K. Adams, J. Goldwaithe, Direct control of a computer from the human central nervous system, IEEE Trans. Rehabil. Eng. 8 (2000) 198–202.
[35] B. Pesaran, J.S. Pezaris, M.S. Sahani, P.P. Mitra, R.A. Andersen, Temporal structure in neuronal activity during working memory in macaque parietal cortex, Nat. Neurosci. 5 (2002) 805–811.
[36] M.D. Serruya, N.G. Hatsopoulos, L. Paminski, M.R. Fellows, J.P. Donoghue, Instant neural control of a movement signal, Nature 416 (2002) 141–142.
[37] J. Wessberg, C.R. Stambaugh, J. Kralik, P.D. Beck, M. Laubach, J.K. Chapin, J. Kim, J. Biggs, M.A. Srinivasan, M.A. Nicoleis, Real-time prediction of hand trajectory by ensembles of cortical neurons in primates, Nature 40 (2000) 361–365.
[38] J.K. Chapin, Neural prosthetic devices for quadriplegia, Curr. Opin. Neurobiol. 13 (2000) 1–5.
[39] J.P. Donoghue, Connecting cortex to machines: recent advances in brain interfaces, Nat. Neurosci. 5 (2002) 1085–1088.
[40] E.E. Fetz, Real-time control of a robotic arm, Nat. Neurosci. 2 (1999) 583–584.
[41] P. Konig, P.F. Verschure, Neurons in action, Science 296 (2002) 1817–1818.
[42] M.A.L. Nicolelis, Actions from thoughts, Nature 409 (2001) 403–407.
[43] D.J. McFarland, D.J. Krusienski, W.A. Sarnacki, J.R. Wolpaw, Emulation of computer mouse control with a noninvasive brain–computer interface, J. Neural Eng. 5 (2008) 101–110.
[44] F. Cincotti, D. Mattia, F. Aloise, S. Bufalari, G. Schalk, G. Oriolo, A. Cherubini, M.G. Marciani, F. Babiloni, Non-invasive brain–computer interface system: towards its application as assistive technology, Brain Res. Bull. 75 (2008) 796–803.
[45] A. Kostov, M. Polak, Parallel man–machine training in development of EEG-based cursor control, IEEE Trans. Rehabil. Eng. 8 (2000) 203–205.
[46] D.J. McFarland, W.A. Sarnacki, J.R. Wolpaw, Electroencephalographic (EEG) control of three-dimensional movement, Soc. Neurosci. Abst. 2008. Available at http://www.abstractsonline.com/viewer/viewAbstract.asp?CKey={CD433551-1B3F-48F0-9FD0-3DE3D157AE87}&MKey={AFEA068D-D012-4520-8E42-10E4D1AF7944}&AKey={3A7DC0B9-D787-44AA-BD08-FA7BB2FE9004}&SKey={FA317E68-3331-4F94-B0D9-6FA70986F1E4} (November 2008) Abstract 778.4.
[47] M.A. Lebedev, J.M. Caramena, J.E. O'Doherty, M. Zacksenhouse, C.S. Henriquez, J.C. Principe, M.A.L. Nicolelis, Cortical ensemble adaptation to represent velocity of an artificial actuator controlled by a brain–machine interface, J. Neurosci. 25 (2005) 4681–4683.
[48] H.K. Kim, S.J. Biggs, D.W. Schloerb, J.M. Carmena, M.A. Lebedev, M.A.L. Nicolelis, M.A. Srinivasan, Continuous shared control for stabilizing reaching and grasping with brain–machine interfaces, IEEE Trans. Biomed. Eng. 53 (2006) 1164–1173.
[49] D.M. Taylor, S.I. Helms Tillery, A.B. Schwartz, Information conveyed through brain-control: cursor vs robot, IEEE Trans. Neural Syst. Rehabil. Eng. 11 (2003) 195–199.
[50] M. Velliste, S. Perel, M.C. Spalding, A.S. Whitford, A.B. Schwartz, Cortical control of a prosthetic arm for self-feeding, Nature 453 (2008) 1098–1101.
[51] J.R. Millan, F. Renkens, J. Mourifio, W. Gerstner, Noninvasive brain-actuated control of a mobile robot by human EEG, IEEE Trans. Biomed. Eng. 51 (2004) 1026–1033.

[52] F. Galan, M. Nuttin, E. Lew, P.W. Ferrez, G. Vanacker, J. Philips, J.R. Millan, A brain-actuated wheelchair: asynchronous and non-invasive brain–computer interfaces for continuous control of robots, Clin. Neurophysiol. 119 (2008) 2159–2169.
[53] D.J. McFarland, J.R. Wolpaw, Brain–computer interface operation of robotic and prosthetic devices, Computer 41 (2008) 48–52.
[54] C.J. Bell, P. Shenoy, R. Chalodhorn, R.P.N. Rao, Control of a humanoid robot by a noninvasive brain–computer interface in humans, J. Neural Eng. 5 (2008) 214–220.
[55] G. Pfurtscheller, C. Guger, G. Muller, G. Krausz, C. Neuper, Brain oscillations control hand orthosis in a tetraplegic, Neurosci. Lett. 292 (2000) 211–214.
[56] K. Choi, A. Cichocki, Control of a wheelchair by motor imagery in real time, in: C. Fyfe, D. Kim, S.-Y. Lee, H. Yin (Eds.), Intelligent Data Engineering and Automated Learning (IDEAL'2008), Springer, Berlin, 2008, pp. 330–337.
[57] E. Singer, Brain games, Technol. Rev. (2008) 82–84 July/August.
[58] H.K. Kim, S. Park, M.A. Srinivasan, Developments in brain–machine interfaces from the perspective of robotics, Hum. Mov. Sci. 28 (2009) 191–203.
[59] G. Schalk, D.J. McFarland, T. Hinterberger, N. Birbaumer, J.R. Wolpaw, BCI2000: a general-purpose brain–computer interface (BCI) system, IEEE Trans. Biomed. Eng. 51 (2004) 1034–1043.
[60] J. Mellinger, G. Schalk, BCI2000: a general-purpose software platform for BCI research, in: G. Dornhege, J.R. Millan, T. Hinterberger, D.J. McFarland, K.-R. Muller (Eds.), Towards Brain–Computer Interfacing, MIT Press, Cambridge, MA, 2007, pp. 581–593.

The Tools Perspective on Software Reverse Engineering: Requirements, Construction, and Evaluation

HOLGER M. KIENLE

Department of Computer Science
University of Victoria, Canada

HAUSI A. MÜLLER

Department of Computer Science
University of Victoria, Canada

Abstract

Software reverse engineering is a subdiscipline of software engineering, striving to provide support for the comprehension of software systems by creating suitable representations of the system in another form or higher level of abstraction. In order to be effective, reverse engineering needs tool support, which provides functionality to extract low-level facts from the systems, to analyze and generate knowledge about the systems, and to visualize that knowledge so that reverse engineers are able to comprehend the aspects of the system that they are interested in effectively.

This chapter explores the issue of building tools for reverse engineering. Since tools are an important part of conducting research in reverse engineering, it is worthwhile to reflect upon the state of tool building with the goal to advance upon it—and thus to advance reverse engineering research as a whole. We tackle this goal by looking at the issue of tools through a set of lenses. The purpose of each lens is to focus on a critical topic for tool building by surveying the current state-of-the-art and identifying challenges that need to be addressed.

In this chapter we discuss three lenses, namely (1) requirements for reverse engineering tools, (2) construction of reverse engineering tools, and (3) evaluation of reverse engineering tools. The first lens identifies a number of generic quality attributes that reverse engineering tools should strive to meet.

ADVANCES IN COMPUTERS, VOL. 79
ISSN: 0065-2458/DOI: 10.1016/S0065-2458(10)79005-7

The second lens approaches tools from the observation that since tool building is a key activity in research, it should be conducted in an effective and rather predictable manner. The third lens looks at the role that tools play in supporting the evaluation of reverse engineering research. While each lens looks at the topic from a different perspective, taken together they provide a holistic picture of tool building in the reverse engineering domain.

1. Introduction and Background 190
1.1. Techniques, Processes, and Tools 192
1.2. Tool Components 194
1.3. Exploring Tools Through a Set of Lenses 198
2. Tool Requirements Lens 199
2.1. Scalability 201
2.2. Interoperability 206
2.3. Customizability 212
2.4. Usability 221
2.5. Adoptability 227
2.6. Requirements of Exchange Formats 236
2.7. Discussion 240
3. Tool Construction Lens 242
3.1. Tool Architecture 243
3.2. Component-Based Tool Development 247
3.3. Model-Driven Tool Development 252
3.4. Discussion 254
4. Tool Evaluation Lens 258
4.1. Evaluation-Driven Tool Building 262
4.2. Theory-Grounded Tool Building 265
4.3. Discussion 268
5. Conclusions 271
References 274

1. Introduction and Background

"Week by week his understanding of his world improves, the white spaces on his map filling up with trails and landmarks."

Hari Kunzru, *The Impressionist*

Broadly speaking, reverse engineering is the process of extracting know-how or knowledge from a human-made artifact [1]. A human-made artifact refers to an

object that embodies knowledge or know-how that was previously discovered or applied by the artifact's creator. An alternative definition is provided by the US Supreme court, who defines reverse engineering as "a fair and honest means of starting with the known product and working backwards to divine the process which aided in its development or manufacture" [2].

Reverse engineering has a long-standing tradition in many areas, most notably in traditional manufacturing (e.g., automotive and medical devices [3]), but also in technology-oriented and information-based industries such as semiconductors, digital media, telecommunications, electronics, and computer software. The latter area is the focus of this chapter. In the literature, the term *software reverse engineering* is used to emphasize that the target is some kind of software (system). Since we are only concerned with software we will simply use reverse engineering when it is clear from the context that we mean software reverse engineering.

In the software domain, Chikofsky and Cross define reverse engineering as "the process of analyzing a subject system to [. . .] create representations of the system in another form or at a higher level of abstraction" [4]. Thus, the reverse engineering process typically starts with lower levels of abstraction (e.g., source code of a high-level programming language) to create higher levels of understanding (e.g, UML class or interaction diagrams). Note that this definition emphasizes that reverse engineering of software produces some kind of knowledge or facts about the software. In contrast, in the manufacturing area the goal of reverse engineering is often to come up with a process that allows to duplicate an existing artifact in the absence of technical drawings, computer models, or other kinds of technical documentation [3].

To thwart a possible misconception, the purpose of software reverse engineering rarely is to develop competing software [4, 5]. Reverse engineering of "foreign" software is much more likely for the purpose of security assessments, uncovering cryptographic techniques, and ensuring interoperability. In this respect, there is a significant difference between reverse engineering in the software and manufacturing domains. While it can be very challenging to duplicate a manufactured artifact (e.g., a fighter plane), it is trivial to duplicate software via copying it. The open source community has demonstrated that the functionality of huge software systems can be "duplicated" by implementing the software from scratch. In fact, many software engineers wish that they could do just that—to scrap their legacy systems and rebuild the same functionality with a cleaner design and different technologies. This leads to the most common purpose of software reverse engineering which is also the focus of this chapter, namely to enable software owners to gain a better understanding of their own software assets so that they are in a position to evolve it.

It is safe to assume that reverse engineering has been performed since the very beginning of software in the form of debugging and disassembly. Academic interest

into reverse engineering started around the mid-1980s with dispersed publications appearing in a number of conferences and journals (e.g., see reprints of articles in Ref. [6])—leading to the emergence of a dedicated community and the first Working Conference on Reverse Engineering (WCRE) in 1993. While it is fair to speak of a dedicated community of reverse engineering researchers [7], one should understand that reverse engineering blends into other research areas such as program comprehension (a.k.a. program understanding) [8], reengineering [6], software maintenance and evolution [9, 10], compiler construction (front-end) [11], software visualization [12], software metrics [13], software modeling [14, 15], and support for collaboration [16, 17].

1.1 Techniques, Processes, and Tools

Academic research in reverse engineering over at least two decades has yielded a broad portfolio of different techniques, processes/methods, and tools. Important techniques include representations of call graphs, impact analysis, identification of abstract data types, clustering, clone detection (a.k.a. code duplication), architecture recovery, redocumentation, cliché recognition, and extraction of business rules. The diversity of reverse engineering techniques is also demonstrated by the software artifacts that these techniques are targeting [17]. Traditionally, techniques have focused on code of high-level programming languages such as Cobol and C—and to a lesser degree assembly—as well as databases (so-called data reverse engineering [18]). There are also dynamic analyses that provide tracing information about the running system [19]. More recently, reverse engineering has broadened its range to include web sites, bug tracking and repository information, and higher level documentation in the form of UML diagrams and natural language texts.

A reverse engineering process gives important guidance on issues such as identifying relevant stakeholders for a reverse engineering project, setting project goals, defining workflows, and selecting appropriate tools and techniques. At the micro-level, reverse engineers follows a workflow that is characterized by three distinct activities: extraction, analysis, and synthesis/visualization. To illustrate these activities, we describe how a reverse engineer would obtain a call graph from the subject system [20].

Extraction: A reverse engineering activity starts with extracting facts from a software system's sources (i.e., the artifacts). For a static call graph, only facts from the source code itself need to be extracted. It is necessary to know the procedures (or methods) as well as the calls made within the procedures. Furthermore, the position of the calls within the source code (e.g., source file name and line number) is often of interest.

Analysis: Certain analyses are performed that use the facts generated in the extraction step. To obtain a call graph, the analysis has to match procedure calls with the corresponding procedure definitions. This matching is not necessarily trivial, for instance, for a call via a function pointer, or a dynamic method call. With this information it is possible to construct a graph structure that represents the call graph.

Synthesis: Results of analyses are presented to the user in an appropriate form. Information typically is presented in a mixture of both textual and graphical form. A call graph is typically rendered with nodes (representing procedures), arcs (representing procedure calls), and node labels (giving the name of the procedure). A reverse engineering tool may show a static rendering of the call graph (e.g., Ciao/CIA [21, 22]) or may offer interactive functionality to explore the graph, for example, via applying layout algorithms (e.g., Rigi [23]).

In the micro-level process, reverse engineers use different comprehension strategies such as bottom-up (i.e., starting from the code and then grouping parts of the program into higher level abstractions), and top-down (i.e., starting with hypotheses driven by knowledge of the program's domain and then mapping them down to the code) [24]. Some reverse engineers try to understand a program systematically in order to gain a global understanding, while others take an as-needed approach, restricting understanding to the parts related to a certain task [25]. The latter approach can be seen as just-in-time comprehension (JITC) [26, 27]; its concept is nicely illustrated by Holt's *law of maximal ignorance*: "Don't learn more than you need to get the job done" [28].

The micro-level process exclusively focuses on the technical perspective and assumes a single, isolated reverse engineer. This is appropriate for a simple reverse engineering project such as a well-defined technical problem or an academic exemplar. In contrast, the macro-level process aims to be more holistic, addressing not only technical, but also business and policy issues and stakeholders [29]. Thus, the activities of the micro-level are driven by the macro-level process, which sets high-level objectives for the overall reverse engineering effort.

Besides techniques and processes, tools are a crucial result of reverse engineering research. Tools are needed to support and validate novel techniques. Typically, reverse engineering techniques require tool support because performing them manually is impractical. Algorithms embodied in a technique may be too complex or cumbersome to perform manually (e.g., creating a call graph manually via code inspection). Also, whenever the target software system changes, the technique needs to be reapplied (e.g., recreating of a call graph whenever the system changes).

As a result, it is expected in the reverse engineering community that a proposed technique is accompanied by a supporting tool and validation that demonstrates the technique's feasibility. While the supporting tool is typically a proof-of-concept

prototype rather than an industrial-strength tool, it should meet certain key requirements such as scalability and usability, or other objective evaluation criteria. The tool is then evaluated against the requirements or criteria with the help of empirical studies (Section 4). But tools are not strictly a vehicle to validate techniques—they are more than that because there is a symbiotic relationship between building of tools and exploring of research ideas [30].

On the one hand, the construction of tools is an important part of reverse engineering research. On the other hand, tool construction is neither simple nor cheap to accomplish [31]. Tool building is costly, requiring significant resources. This is especially the case if the tool has to be robust enough to be used in (industrial) user studies. Sometimes a significant part of the resources of an entire research group are devoted to building, evaluating, and improving a tool.

1.2 Tool Components

The reverse engineering community has developed many reverse engineering tools—prominent examples include Bauhaus [32, 33], Ciao/CIA [21, 22], Columbus [34], GUPRO [35], Moose [30], Rigi [23], PBS [36], and SHriMP [37]. Importantly, most of these reverse engineering tools have a similar software architecture, consisting of several components with standard functionalities (Fig. 1): extractor, analyzer, visualizer, and repository. The extractor, analyzer, and visualizer components reflect the reverse engineering activities of extraction, analysis, and synthesis, respectively (see Section 1.1). In the following, we give a brief overview of the four tool component types.

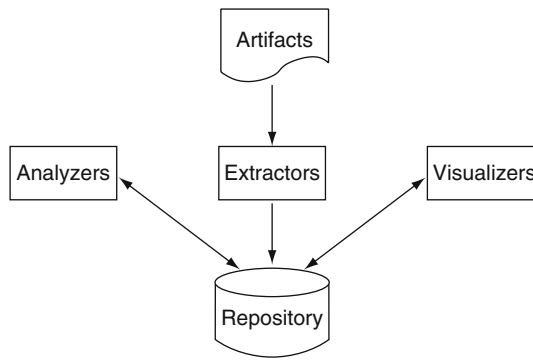

FIG. 1. Components of reverse engineering tools.

1.2.1 Repository

The most central component is the repository. It gets populated with facts extracted from the target software system. Analyses read information from the repository and possibly augment it with further information. Information stored in the repository is presented to the user with visualizers.

Examples of concrete implementations of repositories range from simple text files to commercial databases. Many reverse engineering tools store data in text files and define their own exchange format [38]. For example, the Rigi tool defines the Rigi Standard Format (RSF) [23, 39], which has been adopted by a number of other tools as well. The Moose tool defines the MSE exchange format [40]. Many tool now also support the GXL format [41].

The data in a repository conforms to a *schema*.[1] The purpose of a schema is to impose certain constraints on otherwise unrestricted (data) structures. An important design consideration for a schema is its granularity—it "has to be *detailed* enough to provide the information needed and *coarse grained* enough for comfortable handling" [42].

Schemas are often discussed exclusively in the context of repositories. This is understandable because of the repository's central role to facilitate data exchange. However, the remaining three component types also adhere to schemas, but this is often not recognized because these schemas are typically implicit and internal. For example, extractors, analyzers, and visualizers often use in-memory data structures (such as abstract syntax trees or control-flow graphs). For data export, these components then have to transform the data from their internal representations in order to conform with the repository's schema.

1.2.2 Extractors

Extractors populate the repository with facts obtained from the artifacts that make up the subject system. The extractor has to provide all the facts that are of interest to its clients (i.e., subsequent analyses).

Most reverse engineering tools focus on the extraction of static facts from source code. Static extraction techniques use a number of different approaches. Some extractors use compiler-technology to parse the source. Many parsers are in fact based on compiler frontends. For example, Rigi's C++ parser is built on top of IBM's Visual Age compiler [43]. However, there are also parsers that have been built from scratch such as the Columbus C++ parser [34].

[1] Schemas are also known as meta-models and domain models.

In contrast to parsing, there are lightweight approaches such as lexical extractors, which are based on pattern matching of regular expressions. Lexical approaches are not precise, that is, they can produce fact bases with false positives (i.e., facts that do not exist in the source) and false negatives (i.e., facts that should have been extracted from the source) [44]. On the other hand, they are more flexible than parsers [45]. Typically, lexical extractor tools are agnostic to the target language, and reverse engineers use them to write *ad hoc* patterns to extract information required for a particular task. Such lightweight approaches are a natural match for JITC.

1.2.3 Analyzers

Analyzers query the repository and use the obtained facts to synthesize useful information for reverse engineering tasks. Reverse engineering research has developed a broad spectrum of automated analyses [46, 47].

Analyses can be distinguished by the kinds of representation that they operate upon: lexical, syntactic, control flow, data flow, and semantic [48]. Call graphs, for instance, can be constructed based on lexical matching of function declarations and calls. However, depending on the target language this approach can produce a significant number of false positives and false negatives. For example, in C a lexical approach cannot distinguish between macros and function calls. To reduce such imprecisions, a syntactic approach can be used. To improve precision of the call targets, pointer analysis could be used, which requires the construction of a flow graph.

Another approach to classify analyses is with the help of dichotomies [49, 50]—important ones in the context of reverse engineering are as follows:

Static versus dynamic: Static analyses produce information that is valid for all possible executions, whereas dynamic analyses results are only valid for a specific program run. To assist the reverse engineer, both types of analyses are useful. There are tools that combine both static and dynamic information such as Shimba [51].

Sound versus unsound: Sound analyses produce results that are guaranteed to hold for all program runs. Unsound analyses make no such guarantees. Thus, dynamic analyses are always unsound. Unsound analyses are rather common in the reverse engineering domain. As explained above, many techniques that construct call graphs are unsound because they make simplifying assumptions, thus yielding a potentially wrong or else incomplete result. Still, even if the result of an analysis is unsound, it may give valuable information to a reverse engineer.

Speed versus precision: Static analyses typically trade speed for precision or vice versa. This trade-off is well exemplified by the field of pointer analysis, which has produced various different analysis techniques [52]. Speed is often a problem for analyses that rely on data-flow information, especially if the target systems has millions of lines of code.

Multilanguage versus single-language: Many analyses are tailored toward a specific programming language (e.g., pointer analysis or clustering for C). However, for a (global) analysis to be truly useful in a heterogeneous environment, language boundaries must be crossed. In fact, many industrial systems are based on multiple languages [53].

1.2.4 Visualizers

For software engineers to make the most effective use of the information that is gathered by extractors and augmented by analyzers, the information has to be presented in a suitable form by software visualizers. Mili and Steiner define software visualization as "a representation of computer programs, associated documentation and data, that enhances, simplifies and clarifies the mental representation the software engineer has of the operation of a computer system" [54]. Since the complexity of software makes it difficult to visualize, it is important that tools choose suitable techniques. Particularly, different kinds of visualizations are more or less effective, depending on the comprehension task [55]. Some information needs to be suitably condensed to give a birds-eye view of the subject system (e.g., call dependencies between files or modules rather than procedures), other information has to be presented in the most suitable representation for human consumption (e.g., call dependencies shown in tabular reports or directed graphs).

Many reverse engineering tools use graphs to convey information [56]. The Rigi environment represents software structures as typed, directed graphs and provides a graph visualizer/editor to manipulate the graphs [23]. Rigi graphs can have different colors (to distinguish node and arc types), but all nodes and arcs have the same shape. Polymetric views utilize height and width of nodes, and width of edges to convey additional (e.g., metric) information [57]. The SHriMP visualization tool has a view that represents graphs with a nested layout [58]. Leaf nodes can be opened up to reveal, for example, a text editor (e.g., to view source) or an HTML viewer (e.g., to view Javadoc) [59].

Approaches also differ on how the visualized information can be manipulated. For example, one can distinguish between different levels of modifiability:

Static information: Static information is read-only such as a textual report or a static picture of a graph or diagram. Viewers for such information can offer navigational aids such as hypermedia to navigate between pieces of information [60].

Generation of views from static information: Even if the underlying static information cannot be modified, viewers can offer customizable views to selectively suppress and highlight information. Views can be created by applying filters or layouts. Rigi, for instance, has filters to suppress the rendering of certain nodes and arcs.

Enhancement of static information: Some approaches do not allow the user to change the underlying information, but to enhance it. For example, a viewer might allow the user to add annotations to read-only entities. Rigi allows the user to group nodes into a parent node, facilitating the construction of a superimposed hierarchical tree-structure on the static node structure.

Modification of information: An editor can allow the user to change and manipulate the underlying information. Note that changes are typically constrained in some way in order to ensure the information's consistency. For example, a graph editor that allows the user to delete a node, typically also deletes the arcs attached to that node in order to keep the graph meaningful.

1.3 Exploring Tools Through a Set of Lenses

In a sense, when it comes to tools the "rubber" (i.e., research ideas) "hits the road" (i.e., applying the research ideas). The practicality of a proposed technique can be shown by embodying the technique in a tool and by applying the tool to real-world problems. Conversely, if a tool gets adopted and used in industry it is a strong indication of the usefulness of the tool's techniques and/or process. A tool or prototype is a viable strategy to transition research results from academia to industry.

Given that tools are a crucial part of reverse engineering research that strongly interact with research in reverse engineering techniques and processes, we believe that more emphasis should be directed toward understanding of and improving upon tool building and understanding its overall research impact.

Thus, in this chapter, we specifically focus on reverse engineering tools—as opposed to techniques and processes—and explore their impact on research with the help of three *lenses* that address tool requirements, tool construction, and tool evaluation (Table I).[2] Each lens looks at the topic from a different perspective. Taken together, the lenses provide a holistic picture. For each lens, we describe its

—purpose and *raison d'être*,
—historical importance,
—current state-of-the-art,
—future directions.

The first lens, tool requirements, identifies a number of generic quality attributes that reverse engineering tools should strive to meet. The tool construction lens approaches tools from the observation that since tool building is a key activity in

[2] The idea to approach this topic through several lenses was inspired by the metatriangulation approach as described by Jasperson et al. [61].

TABLE I
LENSES TO EXPOSE REVERSE ENGINEERING RESEARCH CHALLENGES IN TOOL BUILDING

Lens	Explanation	Key topics
Tool requirements	What are the requirements for useful and usable reverse engineering tools?	Quality attributes (scalability, interoperability, customizability, usability, adoptability), functional requirements
Tool construction	How can reverse engineering tools be constructed in an effective and efficient manner?	Component-based tool building, model-based tool building, tool-building process
Tool evaluation	How to evaluate and compare reverse engineering tools? What theories should reverse engineering embrace?	Empirical research, evaluation-driven tool building, theory-grounded tool building

research that is both pervasive and costly, it should be conducted in an effective and rather predictable manner. The tool evaluation lens discusses how tools support the evaluation of research.

In this chapter, we put an emphasis on the first lens, exploring tool requirements in greater detail with the help of an in-depth literature survey. The lenses on tool construction and evaluation are primarily guided by our own experiences and observations. Even though we address the lenses in isolation for clarity, one should be aware that there are interdependencies among them. Where needed, we point out how lenses influence each other.

The rest of this chapter is organized as follows. In Sections 2, 3, and 4, we discuss each of the three tool lenses on requirements, construction, and evaluation, respectively. Section 5 concludes and identifies key issues to advance research in reverse engineering.

2. Tool Requirements Lens

> "Tool research should not be entirely focused on new paradigms, but should address real user needs and expectations."
>
> Kenny Wong [62]

In this section, we discuss requirements—mostly quality attributes—of reverse engineering tools. In order to ensure an objective coverage of the requirements, we have conducted an extensive review of the reverse engineering literature. To our knowledge, this is the first attempt of a comprehensive requirements survey in this

domain. We mainly focus on quality attributes when discussing tool requirements because they equally apply to a broad range of reverse engineering tools with varying functionalities. The identified requirements are useful to communicate assumptions about tool building in the reverse engineering domain. A requirement that has been reported (independently) by several sources is a good indication that reverse engineering tools should meet this requirement in order to be useful and fulfill the expectations of users.

Bass et al. state, "each domain has its own requirements for availability, modifiability, performance, security, or usability" [63]. This is an important observation because it shows that the particular requirements of a certain domain can be better understood by starting from generic, domain-neutral requirements. However, each domain instantiates this generic, high-level requirement differently, depending on the particular domain's characteristics. Thus, it is insufficient to simply state the generic requirements for a domain without further elaborating on them—but that is what most reverse engineering research does. Furthermore, one cannot expect to meet all requirements equally well. In the words of Bass et al., "no quality can be maximized in a system without sacrificing some other quality or qualities" [64, p. 75]. Following this line of thought, a reverse engineering tool or development approach that is missing or lacking in some of the requirements identified in the following subsections can still be satisfactory. However, there needs to be a conscious decision on requirements that is informed by some kind of trade-off analysis. Again, currently research is lacking in this respect.

Even though we chose the term requirements, it is not meant to be interpreted in an overly restrictive sense. In the words of Carney and Wallnau, the term requirements "has connotations that are often overly restrictive" and "the terms 'preference' and 'desire' might be more accurate than 'requirement.'" [65]. Still, since requirements is firmly established in the literature we will stick to it.

Several researchers have discussed requirements of tools in some detail. In his dissertation, Wong has distilled 23 high-level and 13 data requirements for software understanding and reverse engineering tools [62]. Tichelaar discusses six requirements for reengineering environments [66]. Some researchers discuss requirements in the context of a certain kind of tool. For instance, Hamou-Lhadj et al. discuss requirements for trace analysis tools [67], and van Deursen and Kuipers state requirements for document generators [68]. Hainaut et al. analyze requirements for database reverse engineering environments [18]. Other researchers address requirements of specific tool functionalities. Koschke et al. give 14 requirements for intermediate representations in the context of reverse engineering [69], and Ducasse and Tichelaar discuss requirements of exchange formats and schemas [70].

Tool requirements are also exposed by the criteria used in tool assessments, comparisons, and surveys. For example, there are comparisons of the features of

reverse engineering tools [71–75], fact extractors [76, 77], design recovery [78], and software visualizers for static [79, 80] and dynamic code structure [19, 81], and repository-based data [82]. Some of these comparisons are based on the experiences of the researchers assessing the tool(s), while others are based on controlled user studies, structured demonstrations, or questionnaires.

In the remainder of this section, we first discuss the identified quality attributes (i.e., scalability, interoperability, customizability, usability, and adoptability) and then give functional requirements for exchange formats; more details can be found in the first author's dissertation [83]. When discussing quality attributes we first give examples of researchers that have postulated the requirement for reverse engineering tools as a whole. Then, where applicable, we address the implications that a certain requirement has on tool components (i.e., repository, extractor, analyzer, and visualizer), and discuss techniques that enable tools to meet these requirements.

2.1 Scalability

> "Software developers frequently confront issues of bigness, a.k.a scale. A harsh criticism of a solution to a software problem is the comment, 'but it doesn't scale.'"
>
> David West [84]

It is important to realize that reverse engineering tools might be used on subject systems of significant size. For example, one academic reverse engineering tool has been used on Microsoft Excel, which was reported to have 1.2 million lines of C code at the time [85]. A survey among users of software visualization tools has found that there is equal weight on the visualization of small, medium, and large target systems [79]. One-third of the visualized systems were large (i.e., more than one million LOC), one-third were medium (i.e., between one million and 100,000 LOC), and one-third were small (i.e., less than 100,000 LOC). Bellay and Gall have evaluated four reverse engineering tools (Software Refinery, Imagix 4D, Rigi, and SNiFF+) using a 150,000 LOC embedded software system programmed in C and assembly as a case study [72]. Even though this system is well below a million lines, they conclude that "many shortcomings of reverse engineering tools are due to the size of the case study system." Baxter et al., discussing the requirements that the Design Maintenance System (DMS) has to meet, make aware of the relationship between system size and processing time [86]:

> "A fundamental problem is the sheer size of large software systems. DMS is capable of reading literally tens of thousands of source files into a single session to enable analysis

> across entire systems. . . . Size translates into computational costs: 1 microsecond per source line means 2.5 seconds of CPU to do anything to 2.5 million lines."

Whereas some tool implementations handle only limited input, serving often as a proof-of-concept prototype, realistic industrial-strength tools have to handle large target systems. Favre et al. state that "very often, a large number of unexpected problems are discovered when applying good tools at a large scale. This includes not only the size of the software but also the size of the company" [87].

Scalability as a general requirement for reverse engineering tools is discussed by a number of researchers, for instance:

—Brown states for his CodeNavigator tool that it has been designed to "provide information about large-scale software systems" [88]. Furthermore, addressing program understanding tools such as CodeNavigator in general, he observes: "to be effective, they must be able to handle systems of significant size" [88].

—In the context of legacy systems (which are typically large, highly complex, and mission-critical), Wong gives the following tool requirement: "Handle the scale, complexity, and diversity of large software systems" [62].

—Based on their experiences with the Moose reverse engineering environment, Ducasse et al. state that "it should be possible to handle a legacy system of any size without incurring long latency times" [89].

—Lethbridge and Anquetil have developed a list of key requirement for software exploration tools that support JITC. Among the requirements, they state that the system should "be able to automatically process a body of source code of very large size, that is, consisting of at least several million lines of code" [90].

—In his dissertation, Tilley says "it is essential that any approach to reverse engineering be applicable to large systems. For example, effective approaches to program understanding must be applicable to huge, multimillion line software systems" [91].

2.1.1 Repositories

The amount of information that needs to be stored in the repository can affect scalability.[3] Cox and Clark say "a repository is scalable when there are no restricting limitations on the amount of extracted information or code that is stored" [93]. The amount of extracted information can be quite large. For example, a GXL file generated with the Columbus tool to represent the Mozilla 1.6 Web browser has a

[3] This is especially the case for trace data of dynamic analyses, because a small program can, in principle, generate an infinite amount of trace data [93].

size of about 3.5 GB and contains about 4.5 million nodes [94]. The Datrix representation of Vim 6.2 (220,000 LOC) has 3,008,664 relations and 1,852,326 entities [95]. Reverse engineering environments that use exchange formats as their repository should ensure that it "works for multimillion lines of code (e.g., 3–10 MLOC)" [41]. Similarly, Wong requires for a scalable data format to "handle multimillion line software systems" [62]. St-Denis et al. also list scalability among their requirements for model interchange formats ("can be used for real-life, e.g., large-scale applications") [96]. They discuss the following factors that may affect scalability: "the compressibility of the model interchange object, the possibility of exchanging partial or differential models and the possibility of using references (e.g., hyperlinks) to information instead of the information itself." Another factor is incremental construction and loading of information to achieve resource optimization [21, 70]. Hamou-Lhadj et al., discussing the requirements of trace exploration tools, state that input/output performance is critical since traces can become very large and users do not tolerate systems with a poor response time [67]. They also note that "a general XML format, such as GXL, often requires more processing than a tuned special format. Performance of parsing XML poses an obstacle, especially for large data sets."

Analogous to repository scalability, there is a scalability requirement for the in-memory data structures of reverse engineering components. Ducasse and Tichelaar state that "the greater the level of detail, the higher the memory consumption and load time of information from databases or files and the slower the response times of tools that use the information" [70]. For example, the internal representation of Software Refinery's C extractor is about 35 times the size of the source code [97].

2.1.2 Extractors

Since the extractor has to read in and process all relevant sources of the target system, scalability is an important concern. In the context of web service mining, Sneed says that tools "have to be very fast, since they will be scanning through several million lines of code to identify which portions of that code are potential web services" [98]. A related scalability concern is the question "how many models do you need to extract?" since each model might require a unique extractor [70].

As already mentioned in Section 1.2, there is a broad spectrum of extractor techniques with different performance trade-offs. For example, van Deursen and Kuipers observe that "lexical analysis is very efficient. This allows us to analyze large numbers of files in a short time, and also allows us to experiment with different lexical patterns: If a pattern does not yield the correct answer, the analysis can be

easily changed and rerun" [68]. Ferenc et al., discussing the Columbus tool, state "parsing speed" as a requirement for C++ extractors [34]. Bellay and Gall also give "parse speed" among their reverse engineering tool evaluation criteria [72]. They also give incremental parsing as a criterion, which "provides the ability to parse only the parts that changed, thereby reducing the parse time." For performance reasons, extractors are often divided into scanner and parser [86].

2.1.3 Analyses

Researchers seem to discuss the scalability or performance of analyses only if there is a potential problem with the run-time of an analysis. For batch-style analyses, there is a problem if the analysis cannot be executed in a longer time-frame such as a nightly run. In contrast, an analysis' run-time of an interactive tool should be almost instantaneous in the best case. Compared to the other tool components, researchers have stated few general requirements for analyses. An explanation might be that the scalability of analyses is mainly dependent on the time complexity of particular algorithms.

Moise and Wong discuss their experiences with the Rigi tool in an industrial case study. They have developed three specific analyses (written in Tcl) for clustering the subject system according to different criteria. They report for their analyses that "performance is becoming a serious issue with decomposition times running potentially into hours" [99]. Researchers have also discussed scalability and performance issues in the context of trace data compression [19, 67].

An important technique to make analyses more scalable is to "support incremental analyses" [62]. Nierstrasz et al. [30] state for their tool that

> "one key technique that allows Moose to deal with large volumes of data is lazy evaluation combined with caching. For example, although Moose provides for an extensive set of metrics, not all of them are needed for all analyses. Even the relevant metrics are typically not needed for all entities in the system. Thus, instead of computing all metrics for all entities at once, our infrastructure allows for lazy computation of the metrics."

Flexible analyses can trade precision for scalability. For example, Fiutem et al. state for their Architecture Recovery Tool (ART): "To achieve scalability to large industrial size systems, special attention was also devoted to the speed of convergence of the iterative fixpoint method by conceiving a flexible analyzer that allows fine tuning of the trade-off between execution time performance and flow information precision" [100].

2.1.4 Visualizers

A visualizer's rendering speed has to scale up to large amounts of data. This is especially the case for interactive systems that allow direct manipulation of graphs or diagrams. In a user study of visualization tools "users stopped the generation of call graphs because they felt it was taking too long" [101]. Storey et al., reporting on a user study with the SHriMP visualizer, found the following: "The single most important problem with SHriMP views was the slow response of the interface. Since SHriMP views are based on direct manipulation, users expecting immediacy were disturbed by the slow response" [102]. Czeranski et al. made the experience that "the Bauhaus GUI, which is based on Rigi, has a few unpleasant properties. It is relatively slow, which can cause noticeable waiting periods for large graphs, and hence sometimes disrupts the fluent use of the tool" [103]. There is also the issue of lacking performance: "For scalability we must consider if the tool supports large software projects. If the technique does not appear to scale, it may be the implementation which does not scale rather than the technique" [82].

Armstrong and Trudeau state in their tool evaluation that "fast graph loading and drawing is essential to the usability of any visualization tool" [74]. Bellay and Gall use "speed of generation" of textual and graphical reports as an assessment criterion and experienced that "graphical views often need an unacceptable amount of time to be generated because of the layout algorithms" [72]. Contrary to Bellay and Gall's findings, Moise and Wong note that their "case study showed that Rigi can deal with large graphs" [99]. In their survey, Bassil and Keller include "tool performance (execution speed of the tool)" as a practical aspect of software visualization tools [79, p. 5].

2.1.5 Discussion

Most researchers discuss scalability in the context of computational performance and efficiency. However, it is typically discussed without giving explicit or quantitative metrics to measure them. The following is a fairly typical example: "The overall repository software is responsive and quite usable. However, in a few places the performance is slower than we would have liked. For example, large pick lists ($>$500 elements) take a few moments to load" [104].

Considering the large number of publications about reverse engineering tools, there is a surprising low number of more or less quantitative measures about tools' performance such as the following ones:

—"CodeCrawler can visualize at this time ca. 1 million entities. Note that we keep all the entities in memory" [57].

—ISVis can analyze "program event traces numbering over 1,000,000 events" [92].

—"Mozilla compiled on our Linux system 4 in about 35 minutes, while the Acacia extraction took three and a half hours and the translation into TA took another three hours" [105].

Furthermore, as the above examples suggest, each researcher reports measurements based on different criteria and metrics; even tool evaluations tend to not use objective measurements for comparisons among tools [72, 74].

It appears that researchers are also neglecting—or purposely withholding—performance measures. For example, in the domain of trace exploration tools, Hamou-Lhadj and Lethbridge made the following observation [19]:

> "In order to visualize and analyze large program executions, an efficient representation of the event space is needed. Unfortunately, most of the tools mentioned above do not even discuss this aspect, which makes us have doubts regarding their scalability. It is also important to note that most of the experiments that are conducted by the authors of these tools are based on very small execution traces."

There is an increased awareness in the software engineering community that approaches need to scale to industrial demands [106]. As a next step, the development of accepted criteria for the evaluation of tools such as benchmarks [77, 107] would be helpful to make performance and scalability measurements more meaningful.

2.2 Interoperability

> "Building tools is expensive, in terms of both time and resources. An infrastructure for tool interoperability allows tool designers and users to re-purpose tools, which helps to amortize this cost."
>
> Ferenc et al. [108]

Interoperability is the "ability of a collection of communicating entities to (a) share specified information and (b) operate on that information according to an agreed operational semantics" [109]. In other words, tools that interoperate are able "to pass information and control among themselves" [110].

In a small survey about negative aspects of reverse engineering tools, 6 software engineers out of 19 (31%) complained that tools are not integrated and/or are incompatible with each other [90]—this was also the most frequent complaint!

Reverse engineering researchers have recognized the need for interoperability. Woods et al. [111] observe that

—"many tools have been written with closed architectures that hide useful internal products."

—"many external products are not produced in a useful form for further computation." They further conclude that "the bottom-line is that existing program understanding tools and environments have not been designed to interoperate."

Making tools interoperable yields a number of potential benefits. Interoperability enables code reuse in general because it becomes easier for a tool to utilize the functionalities of another one. As a result, reusing of existing functionality can "prevent repetitive 'wheel-creation'" [112]. Zelkowitz and Cuthill view interoperability as a way to improve automation in software engineering: "Tool integration . . . is crucial to effectively provide automated support for activities. In order to automate activities with a tool set, there must be a seamless way to pass information and control among the tools" [113]. Furthermore, interoperability reduces the time and effort to (opportunistically) assemble a tool set that supports a particular reverse engineering task or process, because "no one tool performs well in all tasks" [81]. For instance, the developers of the Dali tool say, that "one of our emphases has been to provide an open, lightweight environment so that tools can be integrated opportunistically" [114].

A number of researchers address interoperability as a general tool requirement; for instance:

—Interoperability is among the tool requirements given by Lethbridge and Anquetil. They require from tools to "wherever possible, be able to interoperate with other software engineering tools" [90].

—Tichelaar, discussing requirements for reengineering environments, states that "a reengineering effort is typically a cooperation of a group of specialized tools. Therefore, a reengineering environment needs to be able to integrate with external tools, either by exchanging information or ideally by supporting run-time integration" [66]. Similarly, Ducasse et al. say, "the environment should be able to operate with external tools like graph drawing tools, diagrammers (e.g., Rational Rose) and parsers" [89].

—Wong addresses interoperability in the context of tool integration [62]: "Tool integration is necessary to combine diverse techniques effectively to meet software understanding needs."

—Interoperability is among the 12 requirements that Hainaut et al. identify for tools that aid in the reverse engineering of database systems: "A CARE tool must easily communicate with the other development tools (e.g., via integration hooks or communications with a common repository)" [18].

—The designers of the Augur tool say, "we would like Augur to be broadly usable in real engineering practice. This means that it must be interoperable with a range of existing tools and infrastructures," and further "Augur's design emphasizes interoperability and extensibility so that it may be incorporated into existing development efforts without significant overhead" [115].

For tools to interoperate, they have to agree on a suitable interoperability mechanism in some form or another. As a consequence, research papers often directly address the question of *how* to achieve interoperability, without explicit stating it as a requirement first.

2.2.1 Techniques

One ambitious approach that has been proposed to achieve interoperability among tools are the Common Object-based Reengineering Unified Model (CORUM) [111] and CORUM II [112] frameworks. These proposals strive to provide a common framework, or middleware architecture, for reverse engineering tools, encompassing standard APIs and schemas for ASTs, CFGs, call graphs, symbol tables, metrics information, execution traces, and so on.

A less ambitious approach compared to the CORUM frameworks is the community's efforts to define a common exchange format. Exchange formats enable interoperability between tools. A particular tool can utilize an exchange format to make information available to another tool. Panas et al. say "in order to have a properly working combination of two or more tools that have been developed independently, they must be able to exchange information. For this purpose we need a common data representation that is general enough to support a variety of data" [116]. Exchange formats use simple files as a (temporary) repository for information transfer. In this case, the coupling between tools that exchange information is loose; an information producer need not to know its information consumers. Examples of exchange formats in the reverse engineering domain are RSF, TA, GraX, CDIF, and GXL; there are also a number of general-purpose data exchange and encoding formats such as ASN.1, SGML, and XML [38].

A file-based exchange format is a rather primitive form of data interoperability because there is no coordination mechanism for concurrent access. Thus, it is the responsibility of the tool user to assure data-consistency and to initiate data transfer from one tool to another. An example of a more sophisticated solution is a repository common to all tools (e.g., in the form of a database management system) such as proposed by the CORUM framework. Whereas a central repository has many benefits (e.g., data consistency), it is also a heavyweight solution. Brown cautions, "the common data repository is often a large, complex resource that must be controlled, managed, and maintained. This can occupy a great deal of time, money, and effort" [117]. Similarly, Wong concludes from the RevEngE project, which used the object-oriented Telos software repository, "there are significant difficulties in using and maintaining advanced integration technologies" [62].

2.2.2 Schema

Exchange formats and common repositories are effective at communicating the structure (or syntax) of data. However, even if tools are able to read the data, it is of little use if they do not know how to interpret it. Schemas are a vehicle to convey semantic information about the data (i.e., its meaning and use). Interoperability among tools is much more effective if they agree on a certain schema [118]. Godfrey puts it this way: "we feel that the particular syntax to define an exchange format is a small issue ... We consider the semantic model (design of the schemas) to be the most important issue" [105]. Whereas syntax is domain-neutral, the schema models a particular domain or reflects an intended use. Thus, a single schema will not suffice. For example, there are schemas with different granularities to represent source code: fine-grained (e.g., a detailed C++ AST [108]), coarse-grained (e.g., the PBS high-level schema to model abstract architectures [119]), and in between (e.g., the Dagstuhl Middle Model [120]). Researchers try to establish standard schemas for other domains as well, for instance, execution traces: "There is also a need for a common [schema] for representing the execution traces of object-oriented systems in order to permit interoperability among tools" [19]. In practice, the diversity of tools makes it difficult to agree on schemas. Moise and Wong make the following observation:

> "Often, an existing schema may not fit as-is to the particular software being analyzed or the tools being used. Consequently, schema reuse is not a simple task, and a proliferation of new schemas has resulted" [118].

All schemas have in common that they have *weak semantics* [121], that is, meaning is derived from the names of schema entities and possibly informal documentation. For example, an entity called *line number* is presumably used on a source code fragment that exists at the given line number in a particular file. However, even if this assumption is correct, it is still not clear if line numbers are counted starting from zero or one, if the line number applies to raw or preprocessed source, if the line number denotes where the fragment begins or ends, and so on.

2.2.3 API

Exchange formats specify the structure of the data and how it is stored on disk. However, how to actually read/write the exchange format and how to represent it in memory is not part of its specification [122]. Thus, tools often implement their own readers and writers. These readers and writers have their own proprietary interface, reflecting the specific needs of a particular tool. Furthermore, the features of the programming language influence the nature and functionality of the API [40, 122].

As a consequence, tools rarely can share an interface and its implementation. Interoperability can be achieved if tools agree on a standardized API to read, write, and manipulate data. A popular example of APIs that enable interoperability for relational data is Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC). The G^{SEE} software exploration tool supports JDBC for data import [123].

Dedicated frameworks for reverse engineering offer an infrastructure for tool integration via common data or control flows. Tools can implement certain interfaces to plug into the infrastructure. The CORUM frameworks are an effort to define common APIs for the reverse engineering domain. However, these efforts have not caught on in the reverse engineering community. Generally, the more sophisticated the interoperability mechanism, the more standardization and agreement among tools is necessary. Jin observes, "although the use of APIs significantly improves the speed and ease of interaction among tools, they still need to know how they can interact with each other. A tool must be aware of the requests it can make of another tool it interfaces with" [124]. This form of tool interaction can be achieved by extending the API with message passing mechanisms based on a message bus (e.g., ToolBus), or point-to-point connections.

One can attempt to draw an analogy between exchange formats (e.g., RSF) and APIs (e.g., ODBC) on how they achieve interoperability. Both abstract from the tools' execution environments. For example, RSF is stored in ASCII files, abstracting away different file systems. Similarly, ODBC abstracts from a concrete relational database. All data adheres to the same structure, or syntax: With RSF, data is represented in tuples, in ODBC data is represented in tables. The underlying model in RSF is a typed, directed, attributed graph; in ODBC it is relational algebra. Both approaches allow schema introspection: RSF has an (optional) schema description file; ODBC has catalog functions.

2.2.4 Discussion

Interoperability and integration of tools has been extensively addressed by researchers in the context of CASE tools and software development environments. Meyers identifies a number of requirements for tool integration: (1) it should be easy to write new tools for use in a development environment, (2) a new tool should be easy to add to the environment without the need to modify other existing tools that are already part of the environment, and (3) there should be no need for different tools to perform the same task (e.g., no more than one parser for a particular language) [125]. He also discusses several approaches to system integration that have been already discussed here: shared file system (i.e., tools exchange data based

on exchange formats), selective broadcasting (i.e., message passing among tools), simple databases (i.e., tools use a common database as repository), and canonical representations (i.e., tools share a common schema).

One approach pursued by the reverse engineering community to achieve interoperability is to agree on a common exchange format. This is exemplified by the thrust to establish GXL. There are also concrete proposals for schemas in a number of areas (e.g., C++ ASTs, mid-level architectural information, and trace extraction), but no proposal has achieved widespread use yet. The community has also considered more ambitious interoperability mechanisms such as a common repository and APIs. For instance, Sim acknowledges the usefulness of a standard exchange format, but argues to move toward a common API:

> "Data interchange in the form of a standard exchange format (SEF) is only a first step towards tool interoperability. Inter-tool communication using files is slow and cumbersome; a better approach would be an application program interface, or API, that allowed tools to communicate with each other directly.... Such an API is a logical next step that builds on the current drive towards a SEF" [126].

However, there is no indication that the reverse engineering community is devoting significant effort to realize this proposal. This is perhaps not surprising, considering that discussion on standard schemas has just begun. In a sense, agreement on an API is comparable to a simultaneous agreement of a standard exchange format along with a number of the most important schemas. Furthermore, exchange formats have their own benefits and it is not clear that APIs are necessarily a move in the right direction.

There is another approach to interoperability that has been mostly ignored so far by the reverse engineering community, namely service-oriented architectures (SOAs). SOA aims to integrate heterogeneous software systems with the use of middleware and services. Software systems expose their functionalities as services, communicating with each other via some kind of middleware. Reverse engineering tools could offer functionalities as web services, and thus allow other tools to discover and call their services in a standardized form. Tool services would allow the development of new reverse engineering functionality via service composition, possibly even on demand. To our knowledge work in this direction is limited so far. Ghezzi and Gall are proposing an approach based on web services [127]. They envision software analysis web services (which are wrappers of already existing tools) that are registered in an analysis catalog (which is organized with the help of a taxonomy). There is an analysis broker that enables users to manage the catalog and to compose services. To facilitate interoperability they propose to use ontologies encoded in OWL to represent inputs to and results of tools.

2.3 Customizability

> "It has been repeatedly shown that no matter how much designers and programmers try to anticipate and provide for users' needs, the effort will always fall short."
>
> Scott Tilley [91]

Customizability is another important requirement for reverse engineering tools.[4] As the introductory quote by Tilley suggests, reverse engineering activities are quite diverse and depend on many factors. As a result, reverse engineers have to continuously adapt their tools to meet changing needs. Thus, it is insufficient for a reverse engineering tool to be general (i.e., it can be used without change in different contexts), it has to be flexible as well (i.e., it can be easily adapted to a new context) [128].

Michaud looks at software customization by asking what is customized, how is it customized, who performs it, and when does it occur [129]. In terms of what is customized one can distinguish between data, presentation, and control customization. In terms of how software is customized one can distinguish between source code customization and other forms that do not require to write code such as option screens, wizards, and configuration files. Customizations can be performed by the tool designers, a dedicated person who customizes the tool to suit a group of users, and the tool users themselves. Customization mechanisms are put in place during tool development by the original tool designers. These mechanisms are then used when installing the tool and during run-time.

In the context of data reverse engineering, Davis observes that "tools need [to be] customized to each project" [130]. Customizability also enables to meet needs that cannot be foreseen by tool developers, for instance, if a tool is applied in a new context or domain. Best states, "if the designer does not create an architecture that lends to extensibility, opportunities to use the tool in other domains can be missed" [131]. Tilley characterizes such customizable tools as domain-retargetable [91]. Conversely, a tool that is not customizable is probably too rigid to meet the changing needs of reverse engineers except in a few well-understood circumstances. Markosian et al. say for reengineering tools, "in our experience, *lack of customizability* is the single most common limiting factor in using tools for software analysis and transformation" [132].

Many tool developers, including commercial ones that produce mass-market software, see customizability as an important requirement to satisfy their customers.

[4] Since most researchers do not distinguish between customizability and extensibility, we use both terms interchangeably. A concept related to customizability is *end-user programmability*, which allows users of an application to tailor it programmatically to their needs [92].

In the following, we give examples of researchers that discuss extensibility as a general tool requirement:

—Buss and Henshaw discuss their experiences with the reverse engineering of IBM's SQL/DS system. Among the lessons learned, they state that "any reverse engineering toolkit must be extensible to meet your problem domain needs," and "since reverse engineering is an open-ended, context-dependent activity, it is imperative that its toolkits be similarly open-ended, flexible, extensible, and versatile" [133].

—In his dissertation about domain-retargetable reverse engineering, Tilley states, "a successful reverse engineering environment should provide a mechanism through which users can extend the system's functionality" [91].

—Bellay and Gall's evaluation framework for reverse engineering tools contains a toolset extensibility criterion: "Tool extensibility is an important feature of many types of tools. This is also the case for reverse engineering tools, in which additional functionality often needs to be integrated to meet the specific constraints of a reverse engineering activity" [72]. The framework further distinguishes between extensibility of parsers, user interfaces, and tool functionality.

—Hainaut et al. give "functional extensibility" for CASE tools as a requirement, motivating it with "no CASE tool can satisfy the needs of all users in any possible situation" [18].

—Reiss has developed a tool, CLIME, to aid software maintenance by formulating constraints on development artifacts such as source code, UML design diagrams, comments, and test cases. Such a tool should be "adaptable to new design techniques and approaches" and as a result "must be open and extensible" [134].

Typically, tools enable customization of their functionalities via configuration files, built-in scripting support, or programmable interfaces [129].

2.3.1 Repositories

A repository consists of a schema and the data stored according to the schema. Each repository provides a rudimentary form of extensibility, because the data that is stored in the repository is not fixed, but customized by the applications that uses the repository. Thus, a more meaningful form of repository customizability is to look at the customizability of a repository's schema. In fact, customizability of a reverse engineering tool is often realized with an extensible schema. The developers of the Moose reengineering tool state, "the extensibility of Moose is inherent to the extensibility of its [schema]" [89].

A number of researchers agree that an exchange format "should be extensible, allowing users to define new schemas for the facts stored in the format as needed" [119, 126]. More specifically, Ducasse et al. require that "an environment for

reverse engineering and reengineering should be extensible in many aspects: . . . the [schema] should be able to represent and manipulate entities other than the ones directly extracted from the source code (e.g., measurements, associations, relationships, etc.)" [89]. Among the requirements that Ferenc et al. have identified for a C++ schema is the need for schemas to "be modular and easily extensible" [34]. Similarly, Riva states that an exchange format "should be easy to extend by the users themselves without the knowledge of complicated procedures" [135]. One of Wong's requirements for a reverse engineering repository is to "support dynamically evolvable schemas" [62]. He further elaborates, "this flexibility to evolve schemas dynamically and incrementally is especially important in software understanding. New needs and concepts often arise over time." Event traces are an example of data obtained with a dynamic analysis. The ISVis tools supports the extension of trace data with new event types without having to change the tool itself: "As far as ISVis is concerned, events have types, and the exact nature of the type is unimportant to the pattern matching ISVis provides" [92].

One can distinguish between the following forms of schema extensibility:

Fixed: Fixed schemas model a certain domain, which is not expected to change. For example, the Bauhaus Resource Graph models the design level of procedural languages [33]. A number of analyses and visualizations have been implemented based on this schema. Consequently, changes in the schema are expected to cause changes in the Bauhaus tool.

Ad hoc: This approach allows to add information in an unstructured way, typically in the form of annotations or extensions (which can range from free comments in natural language to formal assertions) [69]. For example, software engineering environments allow tool-specific decorations of abstract syntax trees [125], and UML allows to attach string tags to entities. Tool is then expected to ignore annotations that they do not know.

Domain-extensible: Domain-extensible schemas provide a core schema describing certain common domain features [136]. The core schema can then be extended. The FAMIX schema of the Moose tool provides a language-independent representation of object-oriented features, the core model, which can be extended with language-specific information [89]. Similarly to FAMIX, the Dagstuhl Middle Model allows extensibility via subclassing [120].

Domain-retargetable: Domain-retargetable schemas are domain-neutral *per se*, allowing the specification of any domain. For instance, the TA exchange format has been used to define schemas for architecture recovery at various levels of abstraction [90, 119, 137]. RSF has schemas to represent higher level software architectures, C++, Java, web sites, C, COBOL, PL/AS, LaTeX, and so on. The Rigi tool is a generic graph viewer that can visualize data that adheres to any RSF schema.

2.3.2 Extractors

There are few customizable extractors for reverse engineering. An early example of a customizable parser is Software Refinery's DIALECT. Newcomb and Markosian report their experiences with the migration of a COBOL payroll system [138]. They give a simple customization example of DIALECT:[5] "the OS/VS compiler used for the payroll system allowed some periods at the end of sentences to be omitted; this syntax had not previously been handled by REFINE/COBOL."

TXL is a source code analysis and transformation tool that allows grammar customizations via so-called agile parsing [139]. In TXL, there is a base grammar for the input language that can be customized with grammar overrides (e.g., to support a particular dialect or analysis task). A typical idiom in TXL programs is to first include the base grammar and then to change selected nonterminals with redefine statements. For example, the left side of Fig. 2 shows a definition of a statement nonterminal for a toy imperative language. In order to introduce a block construct to this toy language, a redefine can be used that extends the nonterminal with a block_statement alternative (right side of Fig. 2).

Whereas parsers typically target only a single language and offer very limited customizations (e.g., via command-line switches), lexical analyzers do not target a particular language and can be extensively customized via pattern specifications. Cox and Clark make the following observation about their lexical extractor:

```
% base grammar

define statement
    [declaration]
  | [if_statement]
  | [while_statement]
end define
```

```
% extending a nonterminal

include ‘‘base grammar’’
redefine statement
    ...
  | [block_statement]
end redefine

define block_statement
  ’begin [statement*] ’end
end define
```

FIG. 2. Example of a grammar override in TXL.

[5] Customizations of parsers is difficult to accomplish because it is necessary to understand the particular parsing technique (e.g., LALR or LL) as well as the grammar itself. Since DIALECT is a LALR(1) parser, it probably needs expert knowledge to actually customize it.

> "Lexical tools are often faster to develop than parser-based tools and, when developed using hierarchical pattern sets, can be easily extended or adapted for novel situations. Extension is performed through the addition of new lexical levels or additional patterns in an existing level" [140].

2.3.3 Analyzers

Most program analyses are fixed in the sense that the reverse engineer cannot turn any knobs to influence the outcome of the analysis (e.g., one cannot trade speed for precision, or vice versa). For instance, there are many clone detection analyses, but few of them can be easily customized to control what constitutes a code clone and what does not. However, flexible analyses can be valuable because it allows the reverse engineer to instruct an analysis to focus its effort on information that is most relevant to a particular reverse engineering task [50].

Jackson and Rinard believe that software analyses should give the engineers more control, for instance, to customize the precision of an analysis: "Engineers need different degrees of precision in different situations, at different points in the program, and for different data structures. Applying a single analysis uniformly across the entire program is therefore counterproductive" [49].

The IntensiVE tools offer a specification language to express structural constraints of Java source code (e.g., to check for design patterns or coding conventions) [141]. Constraints are written in a declarative Prolog-style language, called SOUL, that has predefined predicates that match Java language concepts (e.g., MethodDeclaration and CatchClause). The unification semantics of SOUL can be customized to follow different rules. Figure 3 shows how the unification of predicates can be customized. Atkinson and Griswold have developed a whole-program analysis tool that allows the user to control the precision of its analysis as well as its termination criteria [142]. This avoids wasted resources caused by analyses that are more general than a certain reverse engineering activity actually requires.

2.3.4 Visualizers

In Bassil and Keller's survey on software visualization tools, 45% of the respondents rated the "possibility to customize visualization" as "absolutely essential" [79]. The developers of the Sextant software comprehension tool say, "we conclude that software exploration tools should be extensible to accommodate for domain-specific navigation elements and relationships as needed" [143]. Reiss states that "since we cannot anticipate all visualizations initially, it should be easy to add new graphical objects to the system" [144]. Among the requirements that Favre states for

FIG. 3. Customizing SOUL's unification behavior in the IntensiVE tool.

his G^{SEE} software exploration environment is the need for "customizable exploration" [123].

Reiss has analyzed why software understanding tools are often not used and concludes that "the primary reason was that they failed to address the actual issues that arise in software understanding. In particular, they provided fixed views of a fixed set of information and were not flexible enough" [145]. Even though customizations seem important, Wang et al. say that "existing visualization tools typically do not allow easy extension by new visualization techniques" [146]. They have developed the EVolve software visualization framework, which "is extensible in the sense that it is very easy to integrate new data sources and new kinds of visualizations." A new visualization is realized by extending the EVolve Java framework, which already provides abstractions for bar charts, tables, dot-plots, and so on. Storey et al.'s visualization evaluation framework addresses customizability of tool interactions:

> "Effective interaction to suit particular user needs will normally require a high degree of customization. ... Saving customizations and sharing customizations across team members may also be important" [82].

Storey et al. have evaluated 12 different visualization tools, concluding that Advizor and Xia/Creole have high customizability; VRCS, Palantir, and Jazz have low customizability; and the rest having no support for customizability. Based on this study it appears that there is a number of tools that support customizability, but this feature is not yet pervasive.

Visualization tools support a number of customization mechanisms. In the following we discuss typical ones with examples:

Extensibility hooks: Tools can provide hooks that make it easier to add functionality programmatically. In order to add functionality one has to write code in the tool's implementation language. Tool provides basic hooks via subclassing, design patterns, and APIs. There are also more advanced schemes where tools are implemented as frameworks and have plug-in architectures (Section 3.1).

For example, the EDGE graph editor allows to customize nodes and edges via subclassing from base classes that support default behavior [147]. EDGE also provides callbacks that are invoked for actions such as reading a graph, drawing a node, and deleting an edge. Similarly, Graphviz has an API that allows to manipulate the in-memory representation of graphs as described in Dot [148]. EVolve is a visualization framework implemented in Java. In order to support a new visualization, the Visualization class needs to be subclassed and the class' abstract methods must be implemented [146]. The G^{SEE} tool is implemented as an object-oriented framework dedicated to software exploration and a set of customizable tools that instantiate the framework.

Scripting: Tools can offer a scripting layer to simplify end-user programming. AT&T's graphviz provides an interactive editor, Dotty, which can be customized with a dedicated scripting language, called Lefty [148]. The Rigi graph editor can be scripted with Tcl/Tk [23]. This makes it easy to modify Rigi's user interface to provide new user interactions and to change existing ones (e.g., via adding or modifying drop-down menus and pop-up forms). However, Rigi's graph visualization cannot be easily changed (e.g., the shapes of nodes are fixed). The implementors of the Dali architecture reconstruction tool chose the Rigi tool for its extensibility via scripting:

> "We needed to use a tool that provided both domain specific functionality—to do the necessary graph editing and manipulation—and extensibility, to integrate with the rest of the functionality of Dali. We are currently using Rigi for this purpose, since it provides both the ability to manipulate software models as graphs and extensibility via a control language based on Tcl" [114].

Soft Vision is an open visualization toolkit whose functionality is very similar to Rigi's. The toolkit has a layered architecture consisting of a C++ core to improve performance and a Tcl/Tk layer [149]. The C++ core exposes an API to the scripting layer. Customization is accomplished either via the C++ API or Tcl scripting. The authors made the experience that "for most visualization scenarios imagined by our users, writing (or adapting) a few small Tcl scripts of under 50 lines was enough. This was definitely not the case with other reverse engineering systems we worked with" [149].

Mondrian is a visualization framework that can be scripted with Smalltalk [150]. Since the scripting provides intuitive abstractions new visualization can be defined

```
view := ViewRenderer new.
view nodes: model classes
      using: (Rectangle withBorder width: #NOA; height: #NOM;
                                    liniarColor: #LOC within: model classes).
view edges: model inheritances
      using: (Line from: #superclass to: #subclass).
view layout: TreeLayout new.
view open.
```

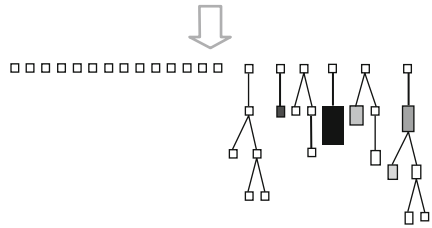

FIG. 4. Scripting a view with Mondrian [150].

with a few lines of code. To give an intuition of Mondrian's capabilities, Fig. 4 shows as sample script and the resulting graph that it generates.

Declarative specifications: Declarative specifications can take various forms, ranging from domain-specific languages to setting option interactively with the tool's user interface. Dot has a specification language to define a graph and to control its rendering. Similarly, EDGE has a description language to specify properties of the graph [147].

The following gives an example:

Graph.x-spacing: 30
Graph.layout.algorithm: ReduceCrossings
Node.borderwidth: 2
Node.label.font: TimesRoman8
Edge.arrow.style: solid
Edge.routing: straight

Configuration files and UI-based customizations can be seen as a rudimentary form of declarative specification.

In the best case, tools offer various customization strategies. For example, the Rigi tool has a startup file to configure fonts, icons, text editor, etc. As mentioned before, more advanced customizations can be performed via Tcl/Tk scripting. However, Rigi's architecture provides no direct support to easily customize its underlying C/C++ implementation.

Having various strategies at their disposal allows users to make trade-offs decisions. For example, programmatic customization is very powerful, but has a

steep learning curve. In contrast, scripting offers better rapid prototyping, but may lack needed performance. And so on.

2.3.5 Discussion

Whereas our survey suggests that researchers see tool customizability as an important requirement, many tools are lacking in this respect, especially extractors and analyzers. In fact, customizability of extractors and analyzers is often forced by scalability problems. For example, dynamic traces easily can become too huge to be efficiently stored, retrieved, and investigated. As a result, trace extractors can be customized for extracting only information at a certain level of granularity (e.g., functions, blocks, or statements), certain event types (e.g., function return or object creation), or parts of the execution (e.g., code of certain classes or packages) [19]. There is also the approach to have a fixed extractor or analysis with the idea to have its result then customized in a separate processing step (e.g., by a subsequent analysis and/or visualization).

There are also analyses that are generic (i.e., they can operate independently of the actual data or schemas) and thus do not need to be customized.[6] A typical example of such an analysis is a graph layout or a textual differencing algorithm. While such generic analyses have the advantage that they are applicable across all schemas, they have the disadvantage that they cannot exploit the domain knowledge encapsulated by a particular schema.

A tool's schema offers a key opportunity to achieve customizability. Indeed, Sim et al. offer the thesis that "tools supporting program comprehension and software maintenance require flexible conceptual models that can be modified as the user or task requires" [151]. Equally importantly, schemas should allow introspection during run-time. This has been already realized by reverse engineering repositories. Similarly to the schemas of repositories, tools should reify conceptual properties such that they can be queried and perhaps even modified during run-time. This could lead to tools that can be customized to the task at hand during run-time (rather than during a dedicated customization step that is performed off-line). A step in this direction is the Fame library, which enables run-time meta-modeling [40].

Last, tools are currently focusing exclusively on customization, disregarding personalization. In contrast to customization, which is controlled by the user,

[6] These analyses typically exploit the fact that all schemas adhere to a common meta-schema (or meta-meta-model). For example, all RSF schemas are composed of nodes, arcs, and attributes, even though concrete schemas differ by the actual types of these entities.

personalization is initiated by the tool itself. It seems worthwhile to try to apply research results of personalization of systems such as web applications to tools.

2.4 Usability

"Nice tool, but it takes some time to get used to it."
user feedback for the sv3D visualization tool [152]

Usability can be defined as the ease of use of a system for a particular class of users carrying out specific tasks in a specific environment [153]. This definition emphasizes that usability depends on the context of use as well as the user.

Usability encompasses a set of other quality attributes or characteristics such as [153, 154]:

Learnability: The system should be easy to learn so that the user can rapidly begin working with the system.

Efficiency: The system should be efficient to use, enabling a user who has learned the system to attain a high level of productivity.

Memorability: The system should be easy to recall, allowing the casual user to return to the system without having to relearn everything.

Satisfaction: The system should be pleasant and comfortable to use.

Among the goals of software engineering are the construction of systems that users find both useful and usable. The same is true for the construction of reverse engineering tools. Meeting the usability requirement of users has several benefits [154]. It improves the product, resulting in productive and satisfied users; increases the reputation of the product and the developer, potentially increasing the product's use; and decreases costly redevelopment caused by user complaints.

Researchers in the reverse engineering field have pointed out the importance of usability as follows:

—In a position statement for a WCRE panel, Müller et al. state, "reverse engineering tool developers not only need to understand the technology side but also need to take the business requirements and the application usability more and more into account" [155].

—Walenstein says in his dissertation about cognitive support in software engineering tools: "The first rule of tool design is to make it useful; making it usable is necessarily second, even though it is a close second" [156].

—Discussing tool design and evaluation, Wong assures that "for program understanding tools, careful design and usability testing of the user interface is important" [62].

—In a talk entitled *Creating Software Engineering Tools That Are Usable, Useful, and Actually Used* Singer states: "Simply put, if a tool isn't usable it won't be used" [157].

—Maccari and Riva have conducted an empirical evaluation of CASE tool usage at Nokia [158]. The respondents rated the modeling requirement to "be intuitive and easy to use" as highly useful (i.e., the median value of the responses was above four on a five-point scale). However, based on their experience with existing CASE tools such as Rational Rose, the respondents replied that this requirement is "insufficiently well implemented."

—In a survey, participants rated the importance of requirements for software visualization tools [79]. The requirement "ease of using the tool (e.g., no cumbersome functionality)" was selected as the second-most important practical aspect, which 72% rated as very important (i.e., the highest value on a four-point scale) [79, p. 8]. However, the authors of the survey believe that "unfortunately, we found a disturbing gap between the high importance attached to the two aspects ease of use and quality of the user interface, and the ratings of these qualities in the software visualization tools in practical use" [79].

Tool developers often are not aware of the importance of usability or do not know how to achieve it. However, in order to systematically identify and improve a tool's usability problems, it is necessary to change tool developers' attitudes toward usability [159]. There are two approaches how one can design for usability [154]: product-oriented and process-oriented.

2.4.1 Product-Oriented Usability

Product-oriented approaches consider usability to be a product characteristic that can be captured with design knowledge embodied in interface guidelines, design heuristics, and usability patterns. For instance, Toleman and Welsh report on the evaluation of a language-based editor's interface based on 437 guidelines [160]. This catalog covers functional areas such as data entry, data display, sequence control, and user guidance [161].

The evaluation of the usability of reverse engineering tools is often focused on the user interface. Toleman and Welsh say, "user interface design guidelines are an important resource that can and should be consulted by software tool designers" [160]. In Bassil and Keller's survey, 69% believe that "quality of user interface (intuitive widgets)" is a very important requirement [79, p. 12]. Reiss, who has implemented many software engineering tools, believes that "having a good user interface is essential to providing a usable tool" [162]. He provides a rationalization for the user interface choices of the CLIME tool, but does not support his decisions with background literature. For the BLOOM software visualizer, Reiss emphasizes

that both usefulness and usability are important: "While it is important to provide a wide range of different analyses and to permit these to be combined in order to do realistic software understanding, it is equally important to offer users an intuitive and easily used interface that lets them select and specify what information is relevant to their particular problem" [145].

Storey's cognitive framework emphasizes usefulness aspects for comprehension tools, but also has a design element that requires to "reduce UI cognitive overhead" [163]. Storey et al. further elaborate on this design element, stating that "poorly designed interfaces will of course induce extra overhead. Available functionality should be visible and relevant and should not impede the more cognitively challenging task of understanding a program" [164].

Design heuristics suggest properties and principles that are believed to have a positive effect on usability. Heuristics address issues such as consistency, task match, memory-load, and error handling. Bass et al. have collected 26 general usability scenarios [165]. A scenario describes an interaction that a user has with the system under consideration from a usability point of view. Examples of scenarios are Aggregating Data (i.e., systems should allow users to select and act upon arbitrary combinations of data), Aggregating Commands (i.e., systems should provide a batch or macro capability to allow users to record, aggregate, and replay commands), Providing Good Help (i.e., systems' help procedures should be context dependent and sufficiently complete to assist users in solving problems), Supporting International Use (i.e., systems should be easily configurable for deployment in multiple cultures), Modifying Interfaces (i.e., system designers should ensure that their user interfaces can be easily modified), and Verifying Resources (i.e., systems should verify that all necessary resources are available before beginning an operation).

To our knowledge, there is no catalog of design heuristics for the reverse engineering domain. However, researchers sometimes relate their experiences, providing tidbits of *ad hoc* usability advice. Examples of such tidbits, grouped by usability characteristics, are:

Learnability: Respondents in Bassil and Keller's survey view learnability as relatively less important; less than half see "ease of learning and installation of the tool" as a very important aspect [79, p. 7].[7] In contrast, Reiss emphasizes that software developers "will use new tools, languages, resources, etc., if (and this is a big if) the cost of learning that tool does not exceed its expected rewards and the tool

[7] Unfortunately, this item in the survey combines two distinct attributes: learnability and ease of installation. The authors decided to group these together because both attributes can be considered as a necessary up-front investment to get productive with the tool (private e-mail correspondence with Rudi Keller).

has been and can easily shown to provide real benefits" [166]. For search tools that are based on pattern matching, Bull et al. require "an easy to specify pattern" [167]. This requirement trades improved query learnability (as well as simplicity and specification time) for less expressive power [168].

Easy installation: A tool should be easy to install. Reiss says that "it is rare to find a software visualization tool that an uninformed programmer can take off the shelf and use on their particular system immediately" [166]. Generally, the more difficult the installation, the less likely that the tool will be tried out. Thus, in the best case a tool requires zero installation. A promising approach is to make tool functionality available via the web browser. D'Ambros et al. say that "if a tool is available as a web application then there is no installation and the cost for people to 'give it a try' is minimal" [169]. The REportal reverse engineering tool is implemented as a Web portal. Users can upload the source code that they want to analyze and then run analyses; thus "users are not required to install any software on their computers in order to use the portal services" [170].

Efficiency: According to Reiss, a tool's usage should "have low overhead and be unintrusive" [134]. Especially, it should not interfere with existing tools or work practices. Reverse engineers often manually inspect, create, and modify the data represented with an exchange format. To simplify this activity, these formats should be human readable and composable (Section 2.6).

Memorability: The scripting interface of a tool should not overwhelm the user with too many commands. For instance, Moise and Wong made the experience that "the Rigi command library was difficult to learn and awkward to use with the sheer number of commands" [99].

Satisfaction: In order to keep reverse engineers motivated, tools should be enjoyable to use. Especially, "do not automate-away enjoyable activities and leave only boring ones" [171]. Tools should be designed to "keep control of the analysis and maintenance in the hands of the [users]" [172]. Otherwise, the user may feel threatened and devalued by the tool.

2.4.2 Process-Oriented Usability

Process-oriented approaches such as user-centered design consider usability as a design goal that can be achieved with a collection of techniques involving the end users (e.g., task analysis, interviews, and user observations). Singer answers the question of "How do you make a tool usable?" with the recommendation to "conduct pilot tests on [the] target user group," and to "bring it to users early and often" [157].

The SHriMP tool has a history of user studies, which also had the goal to improve the tool's usability. For instance, a pilot study (involving 12 users who were

videotaped using think-aloud) has lead to recommendations for interface improvements to SHriMP (and Rigi) [102, 163]. These recommendations were then used to redesign SHriMP's interface [163]. The redesign involved, for instance, a more effective navigation of the visualized graphs combining context+detail with pan+zoom, alternative methods of source code browsing, and the introduction of modes to reduce the cognitive overhead of the users during navigation. The new interface was then evaluated with another user study (which used videotaping and think-aloud, but also a questionnaire and an informal interview) [163, 173]. Besides SHriMP, the TkSee tool is also an example of a reverse engineering tool that has employed user studies (see below).

Awkward usage scenarios or work patterns can also give hints on how to improve usability. For instance, observing the work of professional software engineers, Singer and Lethbridge found that they did "jumping back and forth between tools, primarily Unix command line (performing grep) to editor and back. This jumping involved the use of cut and paste to transfer data and was frequently awkward" [174]. This scenario points toward a better integration and interoperability of tools to improve usability.

2.4.3 Usability in TkSee

The design and evolution of the TkSee search tool is an example of a tool-building effort that has combined both product-oriented and process-oriented approaches in the form of guidelines and user studies to improve the tool's usability. A product-oriented approach was followed by evaluation of TkSee based on Nielsen's usability guidelines [175]. Three evaluators identified 114 usability problems. The types of problems found were poor or missing feedback (e.g., what has happened following an interaction), possible confusion about tool behavior, possible misinterpretation (e.g., meaning of labels or menu items), poor labeling, lack of labeling, lack of consistency, poor graphical design, unnecessary features, lack of needed features, lack of robustness (e.g., tool crashes or hangs), incorrect behavior, and nonoptimal interaction.

The usability analysis of TkSee showed that it is important to have several evaluators with different backgrounds. One of the evaluators had a background in usability, but no background about the problem domain (i.e., program comprehension). This person tended to find general usability problems related to feedback, labeling, and graphical design, etc. In contrast, another evaluator that was already knowledgeable about TkSee and the problem domain tended to point out missing features and incorrect behavior. TkSee was also evaluated with a user analysis involving videotaping and think-aloud usability testing [175]. Eight participants found 72 problems, of which 53% had already been identified before by the

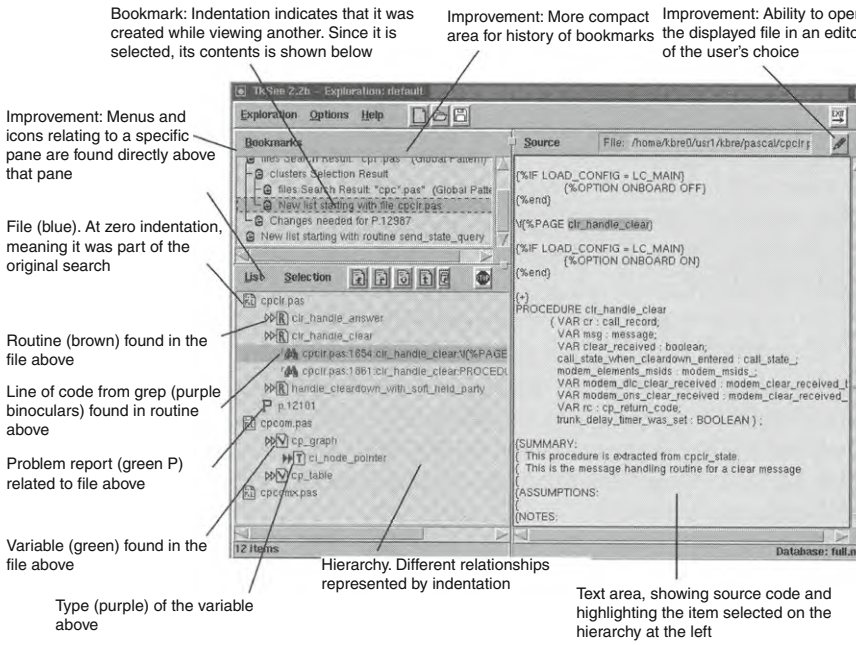

FIG. 5. Changes to TkSee resulting from usability evaluation [175].

evaluators. This shows that both product-oriented and process-oriented approaches are complimentary and that both should be used to evaluate a tool's usability. Figure 5 shows a screenshot of TkSee after its redesign, identifying some changes that were made as a result of the usability evaluation.

2.4.4 Discussion

In 1991, Grudin observed that "resistance to unfriendly systems is growing. There is growing competitive pressure for usability in the marketplace, particularly in mature application domains" [176]. Almost two decades after this statement, it is questionable whether usability of software has improved drastically. Whereas the problem of usability seems to have more visibility, it is still difficult to overcome due to other, competing pressures such as feature creep and time to market.

Usability is recognized as a problem by researchers, but it is often addressed in an *ad hoc* manner. Tool developers rarely discuss how they established the usability of their design. Toleman and Welsh testify, "in general, the design rationales for

software tools that are available rarely indicate the basis for the design of the user interface" [160]. There is the underlying problem that researchers in reverse engineering have neither made an attempt to define nor clarified what they mean by usability. As a result, usability is often judged subjectively by the tool developers. Toleman and Welsh criticize that "software tool designers consider themselves typical users of the tools that they build and tend to subjectively evaluate their products rather than objectively evaluate them using established usability methods" [160]. Too often, usability is only superficially addressed. Lanza and Ducasse address the usability of their CodeCrawler visualization tool by saying that "our tool has been downloaded over 2000 times and, although we have not performed a user survey yet, from personal and e-mail discussions with the users, we have learned that after a short learning time they know what they can get out of each view" [57]. Whereas these indications are encouraging for CodeCrawler indeed, they cannot replace a more formal usability assessment (such as exemplified by SHriMP and TkSee). In contrast, the developers of the Sextant software exploration tool go one step further by first stating five functional requirements (i.e., integrated comprehension, cross-artifact support, explicit representation, extensibility, and traceability), and then arguing how Sextant meets these requirements [177].

Storey's cognitive dimensions framework is mostly focused on improving the usefulness of a program comprehension tool, but not its usability. Green and Petre introduce a framework to assess the usability of programming environments [178]. As a start, researchers should apply existing usability framework while developing usability guidelines specifically for the reverse engineering domain.

2.5 Adoptability

> "Technologists tend to think that if they build a good thing, people will find their way to it and adopt it on their own, based on its inherent goodness...Wrong."
>
> Lauren Heinz [179]

For almost all new ideas, practices, technologies, tools, and other innovations[8] in general, there is the concern of how to get them adopted. As the above quote suggests, adoption of innovations cannot be taken for granted, regardless of the perceived benefits by its proponents. This painful experience has been repeatedly made by different innovators in diverse areas. An example of a famous adoption

[8] Roger defines an innovation as "an idea, practice, or object that is perceived as new by an individual or other unit of adoption" [181, p. 11].

problem in the software engineering area are CASE tools. Even though CASE tools were promoted as significantly increasing software development effectiveness in terms of productivity and quality, many developers did not use these tools or abandoned them later on—leading to questions such as "why are CASE tools not used?" [181].

The attempt to move toward a common exchange format for reverse engineering is an interesting example of the adoption of a new standard within a research community. The goal of a common exchange format is to simplify tool interoperability. However, to achieve this vision, a diverse group of stakeholder have to agree to adopt a standard first. Establishing a standard exchange format or schema is difficult because existing tools have to be modified for compliance, which may not be economical [108]. But without a critical mass, an exchange format does not make the transition to a standard exchange format. Adoption of a new format can be encouraged by addressing the functional and nonfunctional requirements of its users.[9] In other words, the exchange format should be an "early and clear win for adopters" [182].

Perhaps the most import theory that is able to explain adoption is diffusion of innovations [180]. The theory's roots are in sociology, but there are hundreds of publications that have applied it to study the adoption of innovations in a vast number of fields. Diffusion of innovations has identified the following characteristics as most significant [180, p. 15]:

Relative advantage (+): Relative advantage is the degree to which an innovation is perceived to be better than what it supersedes. The immediacy of the rewards of adopting an innovation is also important and explains why preventive innovations have an especially low adoption rate.

Relative advantage can mean that reverse engineers notice that a certain tool allows them to perform certain tasks with more ease, in less time, or with higher job satisfaction.

Compatibility (+): Compatibility denotes the consistency with existing values, past experiences, and needs. Generally, innovations are understood by putting them in relation with the familiar and the old-fashioned.

Favre et al. make the observation that "users are more likely to adopt a tool that works in the same environment they use on a daily basis. This means that SE tools should be integrated to the existing set of tools" [87]. For example, Buss and

[9] Whereas it is clear that the users of an exchange format are researchers in the reverse engineering field, it is difficult to assess the requirements of this rather diverse community. Proposals for standard schemas face the same problem; a group of researchers proposing a schema for C/C++ note, "there is one fundamental issue that we have not yet resolved: who are the end users of this schema and what are their requirements?" [109].

Henshaw report on a platform conflict of their tool; as a result "the product maintainers are uncomfortable with the unfamiliar environment on which the analysis is run" [183]. The tool designers of sgrep try to improve adoption by making their tool compatible with a popular existing search tool, grep. They state, "sgrep is designed to be used in place of grep, so it is important that many of the design decisions found in grep, transfer over to sgrep" [167]. Since grep is a command-line tool, sgrep follows this pattern: "Although graphical user interfaces are often easier to use for novice users, we believe that *familiarity is more important than ease of use*, for the kinds of tasks we envision for sgrep" (emphasis added).

Complexity (−): Complexity is the difficulty of understanding and using an innovation.

Adoption of a tool can be increased by making it easier to use, or by providing training sessions and appropriate documentation. Complexity can also be reduced by identifying and eliminating unnecessary tool features.

Trialability (+): Trialability denotes the degree to which an innovation can be experimented with, without committing to it.

The authors of a work practice study involving the TkSee tool believe that an important factor in the adoption of the tool by developers was that "we allowed them to continue their existing work practices (e.g., use of grep), rather than forcing them to adopt a radical new paradigm" [174]. Also, the TkSee search tool can be easily tried out because reverse engineers can readily switch between TkSee and other search tools that they were using before. A tool is easier to try out if it is easy to install (Section 2.4).

Observability (+): Observability is the degree to which the results of an innovation are visible to others.

For instance, if a tool is visibly used for other developers, it can have a beneficial impact on adoption. Kollmann relates an experience he made in an industrial project: "It can be observed that once a certain number of people have made positive experiences with [an innovation], the propagation is often carried out considerably faster. People seem to trust the experiences others have made and are easier convinced to come aboard, resulting in a kind of snowball effect" [184].

These characteristics help to explain the rate of adoption. Adoptions that are positively related ("+") with the above characteristics will be adopted more rapidly than other innovations. Besides these characteristics, there are other factors that determine adoption, for instance, communication channels, nature of the social system, activities of change agents, and individual/collective decision-making.

Developers of reverse engineering tools have mostly ignored the question of whether their tools are actually adopted by software developers and maintainers. For program understanding tools, Mayrhauser and Vans have observed expectations that users better adapt to a tool if they want to use it: "we still see attitudes reflected in tool builders' minds that if we just teach programmers to understand code the way

they ought to (i.e., the way their tools work), the understanding problem will be solved" [185]. Thus, instead of lowering adoption barriers and increasing the users' incentives to adopt, this attitude expects users to pick up a tool in spite of the raised adoption hurdles.

In the last few years, however, the reverse engineering community has started to pay more attention to this question. This trend is exemplified by the following sources:

—In 1996, Rugaber and Wills already point out that there is an adoption problem of reverse engineering tools: "Reengineering research has had notably little effect on actual software reengineering practice. Most of the published papers in the field present techniques supported by prototype tools; few of which have actually been used on real projects" [186].

—Eight years later, the organizers of the *Fourth International Workshop on Adoption-Centric Software Engineering* come to a similar conclusion: "Research tools in software engineering often fail to be adopted and deployed in industry" [187].

—Lethbridge makes the general observation that "one of the beliefs that motivates software engineering tools builders is, 'if we build it, they will come.' Unfortunately, they often don't come and we wonder why" [188]. Similarly, Wong stresses that it is not enough to devise a new technique or tool and "simply expect practitioners to pick it up and adopt it" [62].

—Software exploration tools use graphical presentation to visualize information about a software system. Even though researchers perceive these tools as valuable for reverse engineering and maintenance tasks, Storey reports that "despite the large number of software visualization tools, few of these tools are used in practice" [164]. Storey et al. use the adoption problem as motivation to propose a framework of cognitive design elements to guide tool design.

—In a roadmap paper for reverse engineering research for the first decade after the year 2000, Müller et al. state that they believe "perhaps the biggest challenge to increase effectiveness of reverse engineering tools is wider adoption; tools can't be effective if they aren't used" [46].

—In his dissertation, Wong demands from researchers to "address the practical issues underlying reverse engineering tool adoption" [62]. Discussing lessons learned from his Reverse Engineering Notebook, he says, "make adoption issues an integral part of reverse engineering tool research and design" [62].

2.5.1 Tool Adoption Research

There are few researchers who see the adoption of their tools as a first-class requirement for their research endeavor. One notable exception is the Adoption-Centric Reverse Engineering (ACRE) project, which explores the adoption problem

of reverse engineering tools. It has initiated a series of four workshop on Adoption-Centric Software Engineering (ACSE 2001–2004) [187]. ACRE addresses the adoption problem with two lenses, cognitive support and interoperability. Since most research tools only support a few selected program understanding or maintenance tasks, reverse engineers typically have to integrate them with other tools to use them effectively. ACRE proposes to investigate the use of data, control, and presentation integration technologies such as XML protocols, the GXL exchange format, Web browsers, ECMAScript, SVG, Eclipse, and web services to make tools more interoperable and thus more adoptable. The other lens, cognitive support, refers to the means by which tools assist the cognitive work of their users (i.e., thinking and reasoning) [156]. Examples of everyday tools that provide some form of cognitive support are shopping lists, address books, and pocket calculators. Without them, certain tasks would have to be performed with an increased cognitive load (e.g., in terms of memorization and computation).

Lethbridge considers adoption using three factors: costs, benefits, and risks of tool use [188]. Potential adopters often do not perform a formal analysis of these factor, relying on their "gut feeling" instead. Examples of costs of use are (c1) purchasing of the tool, (c2) purchasing of extra hardware or support software, (c3) time to install and configure the tool, and (c4) time to learn the tool. Examples of benefits of use are (b1) time saved by the tool, and (b2) value of the increased quality of work done. Examples of risks are (r1) costs are higher than expected, (r2) benefits are less than expected, (r3) unintended negative side effects (e.g., data corruption), (r4) discontinued tool support, and (r5) difficulty to revert to previous work environment. Discussing the factors, Lethbridge says that "in addition to perceiving costs and benefits differently, adopters will more intensively perceive the risks, and the more risks they perceive, the more their perceived benefits must exceed their costs of adoption to take place" [188]. In contrast, tool researchers tend to focus on costs and benefits, ignoring or down-playing the risks. Lethbridge's factors can be used to assess tool-adoption scenarios. For instance, adopting a reverse engineering tool that is build on top of an office suite (as envisioned by ACRE) should have low purchasing costs assuming the office suite is already used (c1 and c2), a simple installation process if the tool is provided as a plug-in (c3), and favorable learning curve resulting in saved time (c4 and b1). On the other hand, updating the office suite might render the plug-in inoperative (r4) and users might become trapped in a certain data format (r5).

Tilley et al. have looked at the adoption of research-off-the-shelf (ROTS) software [189]. They say,

> "in our opinion, adoption is one of the most important, yet perhaps least appreciated, areas of interest in academic computer science circles. . . . Indeed, it can be argued that 'transitionability' as a quality attribute should receive more emphasis in most software projects"

In applied fields such as software engineering, "it may be a measure of success for the results of an academic project to be adopted by an industrial partner and used on a regular basis." However, whereas adoption is a desirable (long-term) goal for a research project, it is not a necessary criterion for success. This is caused by the academic reward structure, which emphasizes publications rather than workable tools. As a result, the adoption of ROTS software is complicated by lacking "completeness (e.g., a partial solution due to an implicit focus on getting 'just enough' done to illustrate the feasibility of a solution, rather than going the 'last mile' to bring the prototype to market)." This is especially the case if the software is the result of a one-person effort produced as part of a Master's thesis or dissertation. Additional complications for adoption are "understandability (e.g., a lack of high-quality documentation)" and "robustness (e.g., an implementation that is not quite ready for prime time)." Huang et al. also look at the relationship between academia and industry: "Both parties know that they have a symbiotic relationship with one another, yet they seem unable to truly understand what each other needs" [190]. Industry has a potential interest in research results that can mature and then be integrated into existing processes to improve software development. To encourage interest from industry, researcher have to solve relevant problems and not to work on problems that are "removed from the current needs of potential users." Also, adoption of tools and techniques by industry could be encouraged with third-party case studies and quantitative data, but these are rarely available for ROTS software [191].

2.5.2 Adoption Factors

Researchers have suggested many potential factors that affect tool adoption, summarized as follows:

Tool: Researchers have mostly focused on factors that can be attributed to the tool itself. To get adopted, tools have to be both useful and usable. Storey says, "although there are many software exploration tools in existence, few of them are successful in industry. This is because many of these tools do not support the right tasks" [163]. Bull et al. state, "in any field, ease of use and adaptability to the tasks at hand are what causes a tool to be adopted" [167]. Wong believes that "light-weight tools that are specialized or adaptable to do a few things very well may be needed for easier technology insertion" [192]. He also says, "by making tools programmable, they can more easily be incorporated into other toolsets, thus easing an adoption issue of tool compatibility" [62]. Tilley et al. suggest that research tools might be more adoptable if they were more understandable, robust, and complete [189]. Devanbu points to inadequate performance of research tools, which are often not intended for large-scale applications [193]. In contrast to most research tools that

require some effort and expertise for installation, popular tools are easily installed or even already preinstalled [194].

User: For adoption, besides the characteristics of the tool, characteristics of the tool users play an important role. When starting to use a tool, users often want positive feedback very quickly. Tilley makes the point that "it is an unfortunate fact that many developers will give a tool only a short window of opportunity to succeed. If they cannot get the tool up and running in 10 min and see real results, without looking at the manual, they will often abandon the tool" [195]. Many users would not even consider a very small trial period no matter of the potential benefits that it promises—if users are happy with their existing tools they do not see the need of trying (yet) another one. Devanbu [193] says,

> "developers are pressured by schedules, and keenly aware of the need to meet cost, schedule or quality requirements. This engenders a conservative bias towards simple, and/or familiar tools, even if somewhat outdated. Builders of complex tools with steep learning curves (even ones promising significant gains) face the daunting hurdle of convincing busy developers to invest training time"

The TkSee tool has collected some experiences with developers at Mitel. Less than half of the developers at Mitel used the tool for significant work over a 2-year period. For users who did not adopt, Lethbridge and Herrera found that "at some point during their learning attempts, many users had concluded that further learning was not worth additional investment of their very limited time" [175]. Interestingly, user that did not adopt tended to state reasons that revealed misconception of the TkSee tool "either because it had not proved rapidly learnable or else because they had found some aspect of it difficult to understand" [175]. TkSee was introduced to the developers without "extensive documentation or training." Lethbridge and Herrera elaborate on this point, "we feel sure that a more proactive training program might have helped increase adoption to some extent; however, we do not feel that more extensive documentation would have helped much—we hardly ever observed anyone look at the existing documentation" [175].

Organization: Often developers do not make the decision to adopt a tool all by themselves because they are constrained by their organization.[10] This is true for industry, which often mandates certain tools, as well as open source projects that assume a certain toolset (e.g., GCC, CVS, and Bugzilla).

From an organization's perspective, "adopting a different toolset or environment discards the hard-earned employees' experience" [196]. Furthermore, "changing

[10] Rogers defines an organization as "a stable system of individuals who work together to achieve common goals through a hierarchy of ranks and a division of labor" [181, p. 375].

tools requires changing processes, which is also an expensive undertaking. As a result, the state of tool adoption in industry is rather static and conservative" [196]. Devanbu also stresses that tools have to integrate smoothly into an existing process: "Software tools work best if their operation is well tuned to existing processes. Thus, a tool that generates paper output may not be helpful in an email-based culture" [193]. Similarly, Wong says "software understanding techniques and tools need to be packaged effectively and made compatible with existing processes, users, and tools" [62].

In a social system such as an organization, the adoption of innovations can be promoted by change agents or innovation champions [180, p. 398]. If such an entity is missing the innovation probably will not get adopted. Lethbridge and Singer have rediscovered this approach for tool adoption of TkSee: "We had significant difficulty introducing new tools. One technique that seems to hold promise is to train a single individual (in our case somebody new to the organization) and have him or her act as a consultant for others" [197].

Whereas the diffusion of innovations theory has developed characteristics that hold across innovations and social systems, there might be also the need to consider domain-specific characteristics. Cordy reports his experiences with the technical, social, cultural, and economical drivers of the Canadian finance industry and financial data processing in general, which he gained during 6 years of project work with Legasys Corporation [172]. In Cordy's experience, resistance to tool adoption is strong because of unhappy past experiences with many inadequate and premature CASE tools. As a result of the pressure of quick, low-risk enhancements of financial applications, for maintainers "only the source is real" [172]. Thus, reverse engineering tools have to present results in terms of source, and not abstract diagrams. Also, robust extractors are needed because "having no answer is completely unacceptable, and programmers will rapidly drop any analyzer that fails to yield answers due to parse errors." The decision to adopt a tool is not made by upper management, but individually by the maintenance programmers and their group manager. In order to convince programmers to adopt a maintenance tool, it is important that they do not feel threatened by it; the workflow of the tool should be such that "all control is left in the hands of the programmer." Cordy says, "this philosophy of assist, don't replace, is the only one that can succeed in the social and management environment of these organizations." Cordy believes that "by studying the maintenance culture of each industrial community, by treating their way of doing things with respect, and by working to understand how our techniques can best be fit into their existing working environment, we can both increase chances of adoption and enhance our own success" [172].

Cost: The adoption of an innovation can be explained by the cost that it causes the users and their organization. Glass [198] says that

> "learning a new tool or technique actually lowers programmer productivity and product quality initially. The eventual benefit is achieved only after this learning curve is overcome. Therefore, it is worth adopting new tools and techniques, but only (a) if their value is seen realistically and (b) if patience is used in measuring benefits."

Patterson makes the point that research software is free of charge, but this does not mean that adoption and use of it has no cost—there is a difference between cost of purchase and the cost of ownership [199]. An example of the cost of ownership is tool administration: "Once installed, some complex tools require a great amount of administration. ... The amount of work and the skills required could be a serious barrier to the adoption of complex tools" [87].

Tilley and Distante say, "ultimately, people will only adopt a technique if they see significant benefit in its use that far outweigh the costs associated with learning and using the technique in the first place" [200]. Storey et al. state that "the adoption of any tool has a cost associated with it. Economic cost is a key concern, in addition to other costs such as the cost of installing the tool, learning how to use it, and the costs incurred during its usage" [82]. In their tool survey, Bassil and Keller did ask for the importance of the "cost of the tool" [79, p. 1]. Interestingly, 50% of the respondents from academia rated this aspect as "very important," compared to only 32% of respondents from industry.

2.5.3 Discussion

There are many factors that influence the adoption of a tool—and many of these cannot be influenced by the tool developers directly. Rifkin reaches a similar conclusion when he says that "as designers of processes and tools that we want adopted by others, we should understand that there is only so much power in the technical content of our processes and tools" [201]. However, tool developers should make an effort by leveraging the factors that they are able to influence to increase the likelihood of tool adoption.

In order to increase the incentives of academic researchers who conduct applied research to focus more on the adoption of their proposed tools and techniques, it is necessary to change the academic reward structure. Researchers already have an incentive to raise adoption to a first-class requirement because an adopted tool has indirectly proven its usefulness [202]. However, as Storey points out, the opposite is not necessarily true: "A lack of adoption is not enough to indicate that a tool is not useful as there are many barriers to adoption (e.g., seemingly trivial usability issues can impede usage of a tool)" [203].

Researchers in the software engineering area have drawn from existing work to understand and improve adoption of tools, technologies, and methods. ACRE draws

from ideas of cognitive science to understand, measure, and evaluate the cognitive support of tools [204]. Storey et al. include cognitive support in their evaluation framework of software visualization tools [82]. Sun and Wong apply cognitive theories of human perception, especially Gestalt theory, to evaluate the SHriMP tool and to suggest improvements for it [205]. Lethbridge et al. incorporates elements of the technology acceptance model and diffusion of innovation theory to explain tool adoption. Tilley et al. look at the adoption of research tools through the lenses of Moore's *Crossing the Chasm* and Christensen's *The Innovator's Dilemma* [189]. Examples of other applicable theories and models are cognitive fit, consumer behavior theory, technology transfer, SEI's Technology Transition Practices, and technology readiness level.

Besides reverse engineering, other computer science areas have discussed adoption of their tools and techniques (e.g., product lines [206], open source [207], groupware [208], and functional programming [209]). There are many more examples, including workshops on technology transfer and adoption (e.g., [210]), and journal special issues (e.g., [211]). It is encouraging that a growing number of researchers have started to realize that adoption is an important challenge that needs to be addressed. Unfortunately, these efforts are still immature. Suggestions to improve adoption are often based on guesswork without providing an underlying theory, or apply existing theories without empirical data.

2.6 Requirements of Exchange Formats

The discussion, so far, has centered on quality attributes. However, the functional requirements of tools are equally important. Indeed, tool builders have to evaluate the functional requirements of their tools carefully, because a tool can be only useful to its users if it provides the functionality that the users need to fulfill their tasks. In fact, the second-largest complaint in Lethbridge's survey was missing or wrong mix of tool features (15%) [90]. As a result, Lethbridge and Anquetil require tools to "incorporate all frequently-used facilities and advantages of tools that software engineers already commonly use" [90]. But what are these frequently used facilities?

In contrast to quality attributes which are more component-agnostic, functional requirements are typically tied to a particular component of the reverse engineering tool (Fig. 1). As an example, we will here discuss the functional requirements for exchange formats, which is an approach to realize the tool's repository component. In other work, we have explored requirements for software visualization tools in more detail [212].

The research community has discussed extensively the functional and nonfunctional requirements for exchange formats. This discussion was started by the realization that a common exchange format would be beneficial for the whole

community.[11] In the following, we briefly summarize the requirements for exchange formats reported in the literature:

Graph model: Wong recommends to "use a graph model with multiple node types, arc types, and attached attributes" [62]. Examples of exchange formats that adhere to this model are RSF, TA, MSE, and GXL.

Version control: One of Wong's data requirements is to "provide version control over schemas and fact bases" [62]. An exchange format does not need to provide version control, but should also not unduly complicate the use of it.

Textual form: An exchange format should be textual. This simplifies processing and makes the format human-readable [96, 214]. Riva notes that "a human readable format allows us to navigate the data with simple text editors and eventually repair corrupted files" [135].

File-based: There are researchers who say data should be stored as files. Riva made the experience with FAMIX that it is convenient when "all the extracted data are contained in one single file that is easy to archive and transfer" [135].

Formality: An exchange format should be well defined and formally documented to "eliminate the possibility of conflicting interpretations of the specification for encoding and decoding model data [96].

Composability: The fact bases of an exchange format should be composable [62]. For example, two RSF files can be simply appended to form a new, syntactically valid RSF file. A related requirement is that the exchange format "should be incremental, so that it is possible to add one subsystem at a time" [41, 119]. Flat formats such as RSF are easier to compose than nested ones such as XML [70].

Granularity: The exchange format should "work for several levels of abstraction" such as fine-grained AST-level and coarse-grained architectural-level [119]. Similarly, Koschke et al. state that an intermediate representation "should support different levels of granularity from fine-grained to coarse-grained" [69]. Kamp says, "the repository should support the use of the *level of granularity* appropriate to the current program comprehension task" [42].

Neutrality: The exchange format should be neutral with respect to the stored information and the platform. For example, it should "work for several source languages" and "work for static and dynamic dependencies" [41, 119]. For St-Denis et al., "the neutrality requirement ensures that the model interchange format is *independent* of user-specific modeling constructs in order to allow a maximum number of model users to share model information" [96].

[11] Examples of discussion forums are WoSEF (held at ICSE 2000) [214], a WCRE 2000 working session on exchange formats, and Dagstuhl Seminar 01041 on *Interoperability of Reengineering Tools* (held in January 2001).

Incremental loading: Ducasse and Tichelaar state that "incremental loading of information is about the ability to load new entities or additional information for entities that already exist in a model. The reasons for considering incremental loading are resource optimization and the merging of information from different sources" [70].

Naming: Entities in the source model have to be represented in a suitable form in the exchange format. This mapping should be unambiguous (e.g., variables with the same name but in different scopes should be distinguishable from each other [62]). This can be accomplished with unique (but artificial) identifiers or a unique naming scheme [70]. In the FAMIX model, "all the entities have a unique name that is built using precise rules" [135].

Querying: Tichelaar et al. state that "a large portion of reengineering is devoted to the search for information. Therefore it should be easy to query the exchange format. Especially, processing by 'standard' file utilities (e.g., grep, sed) and scripting languages (such as Perl, Python) should be easy" [214]. Many exchange formats are associated with dedicated query languages, which have their own requirements [215].

Popularity: Even though not a technical issue, popularity and enthusiastic supporters are an important requirement since an exchange format has to facilitate data exchange between many and diverse (skeptical) stakeholders [41, 96, 119]. In this context, it can be desirable that the exchange format supports industry standards or is a standard itself [214].

Koschke and Sim point out that there are different stakeholders for an exchange format such as tool users and tool builders [213]. The requirements of different stakeholders are not necessarily the same. For instance, a tool user might favor a format that is human-readable, whereas a tool builder is primarily interested in a format that can be parsed easily and efficiently. On the one hand, formality is highly desirable to reduce ambiguity. On the other hand, according to Bosworth "simple, relaxed, sloppily extensible text formats and protocols often work better than complex and efficient binary ones. Because there are no barriers to entry, these are ideal. A bottom-up initiative can quickly form around them and reach a tipping point in terms of adoption" [216]. Perhaps not surprising, the reverse engineering community has primarily focused on technical issues of exchange formats, neglecting to discuss overarching issues such as rationale and economic impacts [217].

2.6.1 Schema

The schema is an important part of the exchange format; consequently, researchers have stated dedicated requirements for it. Several of Wong's data requirements are targeted at schemas. According to him, a schema should "support aggregation,

inheritance, hierarchy, and constraints" [62]. Another important requirement is extensibility. Wong believes that the schema should be "dynamically extensible" [62]. Extensibility can be achieved with inheritance of schema entities, which is supported by several exchange formats (e.g., TA and FAMIX). Inheritance facilitates extensibility because it allows to add new schema entities in a defined manner [70]. Extensibility is highly desirable to support multilanguage tools. Ferenc et al. say, "the schema should be independent of any parsing technology" [108] and Riva argues for a "separation between data and presentation" [135]. A lesson learned from the Moose tool is to "make your [schema] explicit" [30]. Wong also says that the exchange format should "support introspection of schemas" [62]. This means that a tool should be able to query the schema itself and not just the data. To facilitate introspection, a meta-schema is needed (which is either formally or informally defined). Ducasse and Tichelaar introduce meta–meta-models as an axis in their tool design space [70]. They state that while an existing meta-schema has the benefit that it is already predefined and agreed upon by all tool users, it is less flexible and constrains extensibility.[12]

2.6.2 Other Domains

Besides reverse engineering, interchange formats are used in many different domains ranging from networking (e.g., ASN.1) and graphics (e.g., JPEG) to business processes (e.g., PSL) and biosciences (e.g., HDF5). It is an interesting question whether different domains have mostly disjoint or mostly overlapping requirements for exchange formats. If the latter is the case, then different domains can learn from each others' requirements and experiences. There is also the questions if a general exchange format such as XML could replace domain-specific ones.

GraphML is a generic XML-based format to represent graphs and as such its key goal is to "represent arbitrary graphs with arbitrary additional data" [218]. GraphML's designers state as requirements simplicity, generality, extensibility, and robustness. Mendling has looked at exchange format requirements of different domains and found that "the challenges are quite similar across different domains" [217]. Indeed, most of the requirements that he identified apply to the reverse engineering domain: readability, ease of implementation, support of standards, platform independence, efficiency, and free availability. For the schema, he found simplicity in the sense that it is easy to understand, completeness (i.e., to be able to

[12] For example, Rigi's schema is constrained by the fact that it must contain a level arc to model hierarchical graphs. Rigi's meta-schema is constrained because it allows only nodes, arcs, and attributes with a fixed semantics.

represent all relevant concepts of the domain), generality (i.e., to be applicable in all scenarios that are relevant in the domain), unambiguous, and extensible.

Given the overlap of requirements one may ask why the reverse engineering community has pursued the definition of a dedicated exchange format rather than using an existing one. In fact, researchers have leveraged existing formats such CDIF and XML.[13] However, while general exchange formats typically offer generality—because they have been designed with an eye on this particular requirements—and often also extensibility, they are suffering in terms of simplicity and readability. Also, they often lack with respect to ease of implementation, which is a concern if no standard library is available.

2.7 Discussion

Elicitation and documentation of requirements is an important activity during software development. However, requirements are often short-cut or deemphasized when developing reverse engineering tools [220]. For example, researchers often rely on "an intuitive notion of what features are beneficial" when developing a software visualizer [24]. The notion of the tool requirements are thus mostly in the minds of the researchers.

While requirements elicitation is a moving target in a research setting, there can be great value in reifying the tool requirements and their rationale, and tracking changes in the requirements as the tool evolves. One approach to ensure that requirements feature more prominently in tool construction is to employ a dedicated process for tool building, which explicitly address the elicitation and evolution of requirements (Section 3). Tool requirements can also inform tool evaluation (Section 4). For example, one can pick suitable requirements for measuring and comparing tools.

Explicitly articulated tool requirements can play an important role in driving research, especially if a community can agree on a set of desirable and/or idealized requirements whose achievement can serve as a visionary goal. For example, Wong states that addressing all of this 23 tool requirements represents "a significant research challenge," and recommends to "summarize and distill lessons learned from reverse engineering experience to derive requirements for the next generation of tools" [62]. Similarly, Hamou-Lhadj et al. have identified requirements for trace exploration tools and in doing so "have uncovered a number of requirements that raise very interesting research challenges" [67]. The discussed quality attributes

[13] Before the advent of XML, researchers have also proposed to use HTML's meta tags to encode reverse engineering information [220].

reflect the current state of tool requirements and thus can serve as a starting point for future research directions.

In a sense, requirements can drive research from the bottom-up by informing and constraining the tool that gets developed. This approach can complement the more prominent approach of top-down research, which first constructs a tool—based on more or less vague hypotheses or notions—and then tries to find evidence that the tool is useful indeed.

However, most sources of the requirements that we found are based on personal experiences and observations that are inferred, gathered, and reported in an unsystematic manner. In our literature survey, we found that requirements are often discussed without citing related work or disclosing where else the requirement may have come from, or mentioned without giving a detailed explanation or rationalization. Also, researchers do not discuss the applicable scope of a stated requirement. For instance, is the requirement believed to apply to all software systems, to the domain of software or reverse engineering, to certain kinds of tools, or to one tool in particular? Lastly, a requirement is often discussed in isolation, without addressing dependencies or trade-offs with other requirements. Perhaps surprisingly, only Lethbridge and Anquetil explicitly separate their requirements into functional and nonfunctional ones [90]. Such a distinction makes it easier to judge the scope and applicability of requirements. Furthermore, they explicitly identify requirements that their tool does not address (yet), thus making their tool's limitations more explicit.

In addition to the requirements gathered in an *ad hoc* manner, more formal techniques are needed that are grounded in studying of actual users in a realistic setting; this trend can be already observed in software visualization research [16, 221].

Whereas tool requirements seem relatively stable, they are not fixed. Changes in development and maintenance processes and in the characteristics of software need to be reflected in the requirements for reverse engineering tools. Thus, researchers should continuously reevaluate their assumptions. For example, a previously neglected requirement that is starting to receive more attention by researchers is collaboration and multiuser support [17]. This need was already articulated in 1990 for software development tools [221], and reiterated later on for the reverse engineering domain (e.g., Rugaber in 1996 [48] and Storey in 2005 [203]). In contrast, Bellay and Gall argue that multiuser support in reverse engineering tools is "not of such importance as in development tools because the application normally does not change and only one person may reverse engineer it" [72]. However, this view seems dated considering the large amount of commercial and open source software that is developed and maintained in a distributed, collaborative work-style. Indeed, Koschke states that "large maintenance and reverse engineering projects require

team-work and, hence, visualizations need to support multiple users that may work at the same system at the same time at possibly different locations" [222].

The emerging importance of this requirement is also reflected in Storey et al.'s software visualizer framework, which has a dimension to distinguish the team size that a particular tool targets [82]. Examples of tools that support teams in (near) real-time are the Jazz collaborative development environment [223] and the Churrasco collaborative software evolution analysis tool [224]. There are also commercial IDEs emerging that emphasize collaboration such as IBM's Jazz (jazz.net).

3. Tool Construction Lens

The building of tools is an important part of many research efforts, especially in the reverse engineering domain. The tangible results of reverse engineering research is often embodied in tools, for instance, as a reference or proof-of-concept implementation.

Even though tool building is a popular technique to validate research, it is neither simple nor cheap to accomplish. Tool building is costly, requiring significant resources. This is especially the case if the tool has to be robust enough to be used in (industrial) user studies. Nierstrasz et al., who have developed the well-known Moose tool [30], say that

> "in the end, the research process is not about building tools, but about exploring ideas. In the context of reengineering research, however, one must build tools to explore ideas. Crafting a tool requires engineering expertise and effort, which consumes valuable research resources."

Sometimes a significant part of the resources of an entire research group are devoted to building, evaluating, and improving a tool. Given the significant cost associated with tool building, researchers should explore how tools can be constructed in effective and efficient manner. Wasted resources for tool building translate to less research output, slower iteration and communication of research, and reduced opportunities for adoption and transition of tools and techniques to industry.

Traditional tool building that constructs everything from scratch offers the most flexibility since almost all functionality is implemented from scratch and under the full control of the developer. On the downside, this approach is costly (e.g., in terms of longer development time) and can result in idiosyncratic tools that are difficult to learn and use. On the other hand, as for any software development project, there is the desire in tool construction to reuse. This point is articulated by Shaw as follows [225]:

> "Most applications devote less than 10% of their code to the overt function of the system; the other 90% goes into system or administrative code: input and output; user interfaces, text editing, basic graphics, and standard dialogs; communication; data validation and audit trails; basic definitions for the domain such as mathematical or statistical libraries; and so on. It would be very desirable to compose the 90% from standard parts."

There is a desire among researchers to build upon existing functionality and infrastructure. Participants of a tool building workshop for reverse engineering articulated that they "were tired of writing parsers/analyzers and wanted to avoid writing another one, in particular a C++ parser" [213]. This desire is driven by the realization that in tool building comparably little effort is spent on the research contribution, and that a significant effort is needed for the supporting infrastructure. Researchers would rather work on core activities that advance research than being tied up in lower level plumbing.

There are many different forms of reuse ranging from design and code scavenging to very high-level languages [226]. However, one can distinguish between two major techniques to achieve reuse: compositional and generative reuse. These two reuse techniques correspond to component-based (Section 3.2) and model-driven (Section 3.3) tool development, respectively. Before turning to these two techniques, we discuss an overarching issue, namely architectures for reverse and software engineering tools (Section 3.1).

3.1 Tool Architecture

The importance of architecture on software systems is firmly established [64]—tool construction is no exception in this respect. Importantly, tool architecture interacts with tool requirements (Section 2). The required quality attributes of a system often drive the decision to select a particular architecture or architectural style [63]. Conversely, the chosen architecture of a software system has a profound impact on its quality attributes [227].

When building a tool, fundamental questions of the architecture are how to separate the tool's overall functionality into functional units, and how to interface these units with each other. For example, Fig. 1 shows a generic, high-level reverse engineering architecture, consisting of four components: extractors, analyzers, visualizers, and a repository. This architecture exposes the *conceptual*, *or logical*, *structure* of the software, in which the components (or units) are abstractions of the systems' functional requirements and are related by the shares-data-with relation [64].

One can also look at the introduced conceptual architecture as a *reference model* for reverse engineering tools [64, p. 25]. A reference model emerges through increasing consensus of a research community and thus indicates a maturation of the research domain. The domain of compilers provides an example of a widely known reference model with the following functional units [11]: lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generator, code optimizer, code generator, symbol-table manager, and error handler. Reference models in reverse engineering are important because they provide a frame of reference to guide researchers in understanding and implementing tools in the reverse engineering domain.

The concrete architecture of a reverse engineering tool does not necessarily coincide with the conceptual tool architecture presented above. At one extreme, one could imagine a monolithic architecture that groups extraction, analysis, and visualization into a single component (without any interfaces). In practice, one can distinguish three different architectural approaches for tool building [87, 110, 228]:

Data-driven: In this approach, the tool's (functional) units are rather loosely coupled and communicate via an agreed-upon data model. Data communication can be accomplished with a repository or an exchange/document format.

A typical example of a data-driven integration framework is the Unix *pipe* mechanism, which composes new programs from existing ones (also called *filters*) by connecting the textual output of one program to the input of another.[14] In this case, the components are executable programs and the data model are textual streams grouped into lines and lines grouped into fields via special characters such as whitespaces or colons.

Ciao/CIA is an example of a reverse engineering tool that follows this approach, which the authors call repository-based reverse engineering [22]. A typical use of Ciao is a pipeline that consists of a sequence of query commands followed by a visualization command. Figure 6 shows the typical constituents of such a pipeline [230, p. 188].

Control-driven: In this approach, tool components are more tightly coupled because they are based on an infrastructure that allows them to pass messages among each other (e.g., via a message server). The underlying communication infrastructure can be provided by the operating system or by more sophisticated wiring standards such as CORBA, COM, or JavaBeans.

[14] Salus summarizes the philosophy of Unix as follows: (1) write programs that do one thing and do it well; (2) write programs to work together; and (3) write programs that handle text streams, because that is a universal interface [230, p. 53].

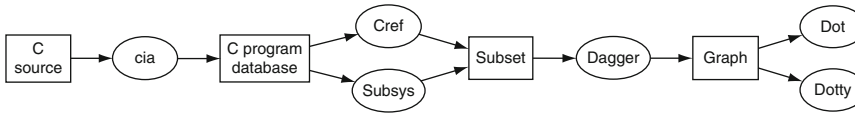

FIG. 6. CIA pipeline [230].

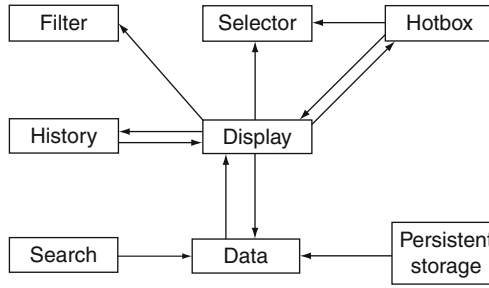

FIG. 7. ShriMP's architecture in terms of JavaBean components (boxes) and adapters between components (arrows) [231].

For example, the SHriMP tool's core architecture now consists of a number of JavaBean components (Fig. 7) [231]. All components know the abstract concepts of the data model, consisting of entities and relationships. Most important are the Persistent Storage, Data, and Display components. The Persistent Storage Bean reads and writes data (i.e., entities and relations) to and from a repository. It passes the data to a Data Bean, which constructs the data's in-memory representation. The Data Bean's interface allows the data to be queried and manipulated. SHriMP already provides implementations of Persistent Storage Beans that read and write RSF and GXL. A generic Data Bean implementation is also provided. Entities and relationships in the Data Bean are visualized as nodes and arcs of a directed, hierarchical graph in the Display Bean. Examples of other components are Search Bean (to search through information associated with entities and relationships) and Filter Bean (to determine whether certain entities and relationships should be hidden). All components are independent from each other and can be replaced with other compatible beans.

A variant of the control-driven approach restricts communication patterns by having a single master component that controls a number of slave components. An example is the use of Emacs as a central component that invokes other services such as compiler, make, lint, and debugger. Many IDEs have been developed on top of Emacs; in fact, XEmacs (then called Lucid Emacs) was created to realize Lucid's Energize C/C++ development environment.

Presentation-driven: This approach refers to a seamless interoperation at the user-interface level. Components are tightly integrated and have a common look-and-feel. Examples of technologies are compound documents such as OLE and OpenDoc. In order to achieve such tight integration some kind of dedicated infrastructure is necessary.

IDEs are a prime example of presentation-driven development tools. Since these IDEs are often customizable based on a plug-in mechanism (e.g., Eclipse, IBM VisualAge, and Microsoft Visual Studio), many reverse engineering tools are implemented as IDE integrations. However, there are also examples of presentation-driven reverse engineering tools that are not based on IDEs (e.g., SolidFX [232] and Columbus) or leverage Web browsers for the user interface (e.g., Churrasco and SPO [169]).

VizzAnalyzer is a plug-in-based tool framework that allows to write analyses and visualization plug-ins [116]. The plug-ins are communicating by manipulating a graph data structure. Figure 8 shows the framework's architecture with two analysis plug-ins (Recorder and Analyzer) and two visualization plug-ins (yEd and Vizz3d). Two of these plug-ins enable to add plug-ins themselves (e.g., to support new layouts for Vizz3d).

The three approaches introduced above are usually inclusive. The presentation-driven approach needs functionality to pass control and data in order to achieve seamless integration of components; and the control-driven approach needs to pass data along.

In practice, tools combine the above approaches. Reverse engineering tools often decouple the extractor component from the rest of the tool with a data-driven approach involving an exchange format.[15] Analyses and visualization are often tightly integrated with a control-driven and/or presentation-driven approach.

The SolidFX tool is based on presentation-driven integration and thus has the look-and-feel of an IDE [232]. The individual components for fact extraction, analysis, and visualization communicate via a central fact database. In order to accommodate third-party tools, SolidFX offers control-integration with a query API to the fact database and data-integration based on various formats (XML for ASTs, XMI for UML diagrams, SQL for metrics, Dot and VCG for graphs). Interestingly, SolidFX's developers previously used the same components of their tool in a loosely

[15] Historically, reverse engineering tools developed in the late 1980s and early 1990s, supported only a single programming language (e.g., MasterScope for Lisp, FAST for Fortran, and Cscope for C [22]). Since these tools consisted of a single extractor, there was often a tight coupling between the extractor and the rest of the system [22, 234]. This rather tight coupling was inflexible and made it difficult or impossible to support additional languages.

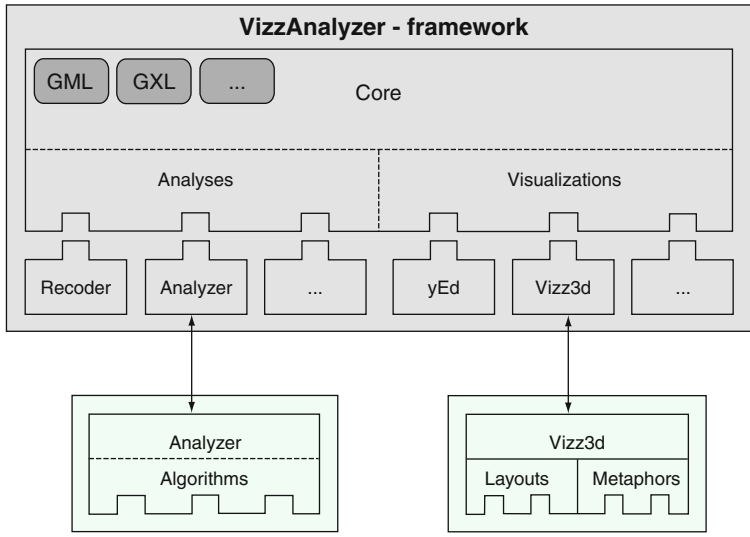

FIG. 8. Two-level plug-in architecture of the VizzAnalyzer Framework [116].

coupled manner without having a presentation-driven IDE. They relate their experiences of these two different architectures as follows:

> "For the parsing phase, the [IDE] was not much more effective—a simple text makefile-like project was sufficient. However, for the exploration phase, the [IDE] and its tight tool integration were massively more productive than using the same tools standalone, connected by little scripts and data files"
>
> Telea and Voinea [232].

3.2 Component-Based Tool Development

> "Programs these days are like any other assemblage—films, language, music, art, architecture, writing, academic papers even—a careful collection of preexisting and new components."
>
> Biddle, Martin, and Noble [234]

Component-based development (CBD) is a widely applied and highly successful approach for developing software systems [235]. Consequently, researchers have started to adopt the idea of CBD for developing their research tools. In the following, we refer to this approach to tool building—which reuses existing, prepackaged functionality—as *component-based tool development* (CBTD). As opposed to

traditional tool building, which is characterized by a high degree of custom code and little reuse, CBTD leverages software components as building blocks. For example, Fig. 9 shows the reuse of components in the CIA tool [230, p. 178]. External components are shown as diamonds, while tool-internal components are shown as ovals.

A key driver for CBTD is the reuse of existing code in the form of components. If a component is carefully selected, its existing functionality can cover a significant part of the tool functionality. As a result, significantly less code needs to be written and subsequently maintained. Reiss has implemented a software development environment, Desert, based on FrameMaker. He reports that for Desert's editor, "FrameMaker provides many of the baseline features we needed," specifically "it displays both pictures and text" and "it can display high-quality program views" [236]. Similarly, the authors of SLEUTH say that "FrameMaker provides an effective starting point for our prototype. Many of the basic features necessary for document creation and editing are provided, allowing effort to be concentrated on more specialized features" [237].

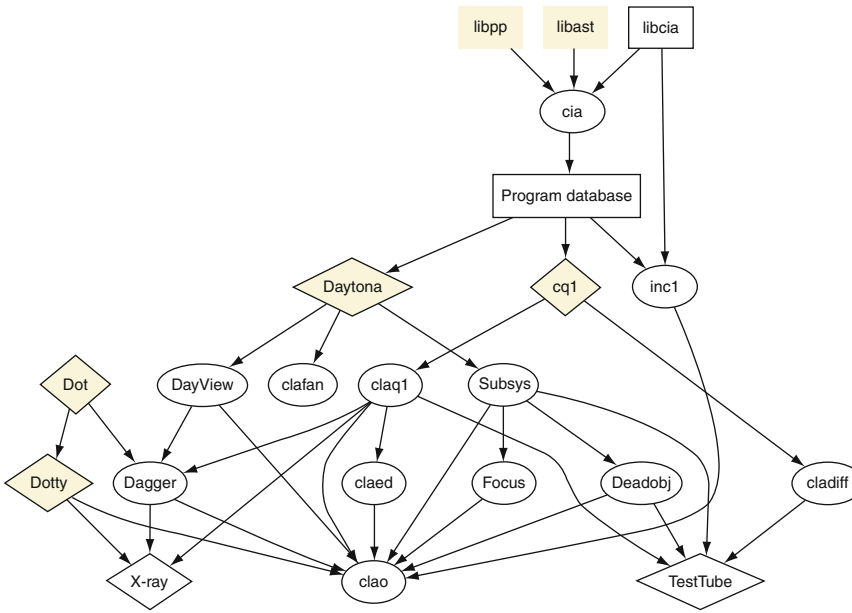

FIG. 9. Reuse of components in the CIA tool.

In principle, all kinds of components are candidates for CBTD. Thus, the definition of a component in the context of CBTD should be rather broad such as the one provided by Czarnecki and Eisenecker, who define (software) components as "building blocks from which different software systems can be composed" [238]. Examples of applicable components for CBTD are

—IDEs (e.g., Eclipse, Microsoft Visual Studio, IBM Visual Age, TogetherSoft Together).

—Commercial off-the-shelf products (e.g., Microsoft Office and Internet Explorer) and their open-source counterparts (e.g., OpenOffice and Firefox).[16]

—Components based on Sun's JavaBeans and Java Enterprise Beans (EJB), and Microsoft's Component Object Model (COM) and Distributed COM (DCOM).

—Object-oriented frameworks to realize GUIs (e.g., Java's Abstract Windows Toolkit (AWT) and Swing, Eclipse's Standard Widget Toolkit (SWT), and the Microsoft Foundation Classes (MFC)).

—Unix tools (e.g., awk, sed, and grep), text editors (e.g., Emacs), and scripting languages (e.g., Perl).

—Libraries for domains such as standard data structures (e.g., C++ Standard Template Library (STL)), graph data structures (e.g., LEDA), as well as library collections (e.g., from AT&T [230]).

The fact that CBTD is widely applied is exemplified by many tools that leverage components to implement functionalities for fact extraction and visualization [83, 235]. Table II gives examples of software and reverse engineering tools that base their visualizations on components. This list represent only a smaller sample, but illustrates the broad range of components that have been leveraged. Finding suitable components for implementing a tool is important because the characteristics of the component will determine the whole tool building effort.

The use of components fundamentally changes the development of tools and has unique benefits and drawbacks. This fact is often not realized by tool builders. Important questions for CBTD that have to be addressed by researchers are: What are good candidate components for CBTD given a tool's application domain, its required functionality, its desired quality attributes, and its envisioned users? What impact has a certain component on the overall tool architecture, on the ramp-up time and implementation effort for the tool, and on the further maintenance and evolution of the tool? What characteristics of the components and the architecture minimize risks and maximize effects? And so on.

[16] Both commercial and open-source products are summarized as off-the-self (OTS) products.

TABLE II
EXAMPLES OF COMPONENTS TO BUILD GRAPH-BASED VISUALIZERS [235]

Component type	Host component	Tool-building examples
OTS products	Office/Visio	REVisio [239]
		Huang et al. [240]
		Nimeta [241]
		VDE [242]
		Galileo/Nova [244–246]
	FrameMaker	SLEUTH [237]
		Desert [246]
	Web browsers	REPortal [170]
		Software Bookshelf [36]
		TypeExplorer [68]
IDEs	Eclipse	SHriMP [247]
		MARPLE [248]
	Rational Rose	Rose/Architect [249]
		UML/Analyzer [250]
		Berenbach [251]
	Together	JaVis [252]
		GoVisual [253]
Tools	AT&T Graphviz	Reflexion model viewer [85]
		ReWeb [254]
		CANTO [255]
	SVG	SVG graph editor [256]
		SPO [257]
Libraries	OpenGL	Extravis [258]
		CodeCity [259]
		SolidFX [232]

The use of components also has an impact on the tool's quality attributes. In the following, we briefly give examples how CBTD can impact the five quality attributes introduced in Section 2:

Scalability: Scalability is especially a concern for visualizations. Generally, components such as Visio and PowerPoint are able to handle the rendering of dozens of graphical objects. This is sufficient for small to medium software graphs. However, the rendering of larger graphs can cause problems in terms of screen updating and rendering speed. For example, the developers of the Nova tool state that their tool's "response time is largely dependent on package performance. ... We spent significant effort understanding Visio's drawing speed and exploring ways to use Visio that would give us better drawing performance" [243]. Also, different components and different versions of the same components can differ in their performance characteristics. For example, in Visio 4.1 the drawing speed of objects

increases quadratically with the number of shapes already present on the page; in contrast, Visio 5.0's behavior is linear [243].

Interoperability: Components can have a wide range of interoperability mechanisms and make implicit assumptions about the way they interact with other components. Thus, it is unlikely that two independently developed components that do not adhere to the same component model will interoperate seamlessly out-of-the-box. This phenomenon is known as *architectural mismatch* [260]. Interoperability is less of an issue if the tool is based on a single component only. Also, architectural mismatch is less of a problem for data integration compared to control and presentation integration. Wrappers and bridges can be used to integrate heterogeneous components; however, this solution can turn out to be brittle. The SHriMP tool is using an AWT-SWT bridge to integrate with Eclipse, but this approach has several undesirable effects (e.g., risk of deadlocks and lost keyboard events) [261].

Customization: Customizations of components are limited by the functionality that the API and/or scripting language provides. Based on their customization experiences with several commercial components, Egyed and Balzer say that "sadly, 'real world' COTS tools are often only partially accessible and customizable, greatly limiting their reuse" [262]. Reiss reports the following limitation in the FrameMaker API: "While FrameMaker notified the API that a command was over, it did not provide any information about what the command did or what was changed" [246]. As a consequence, customizations of components without the use of source code modifications always run the risk that certain desirable tool functionalities cannot be realized at all, or with less fidelity. Nova's developers had to cope with undocumented restrictions of Microsoft Office components: "Due to [the] lack of package documentation, we discovered certain limitations of the packages only after working with them extensively. In some cases, the limitations were quite serious" [243]. For example, they found out that "the maximum number of shapes Visio 4.0 could store on a single drawing page was approximately 5400."

Usability: Familiarity of the target users with a certain product improves usability and helps adoptability. Reiss explains the decision to use FrameMaker as a component with the fact that "we wanted an editor that programmers would actually use. This meant that the base editor must be familiar to them, preferably one they were using already" [236]. Furthermore, popular components can reduce the need for documentation: "Because such components are also popular stand-alone applications, users are often already familiar with them, and much of the application documentation applies when the application is used as a component" [244]. Tool implementors can also leverage the existing infrastructure of the help system to seamlessly provide documentation for their tool. For example, Microsoft Office's Assistant can be customized in Visual Basic to provide tool-specific help.

Adoptability: The market share of components can improve a tool's adoptability. The popularity of PowerPoint was an important factor for the developers of VDE: "Visio is a commercial product with many similarities to PowerPoint . . . It might provide a better technical fit to our needs, but lacks PowerPoint's enormous user base" [242].

3.3 Model-Driven Tool Development

> "Truly model-driven development uses automated transformations in a manner similar to the way a pure coding approach uses compilers."
>
> Kelly and Tolvanen [263]

CBTD reuses existing components, integrating and customizing them so that they fit together. In contrast, *model-driven tool development* (MDTD) generates code from a domain-specific, higher level specification [238, 264]. Thus, MDTD can be seen as a form of generative reuse. The reusable assets in these techniques are less intuitive and less palpable compared to compositional reuse because they consist of patterns for code generation and transformations.

The idea of MDTD derives from generative development approaches such as domain-specific modeling (DSM), model-driven development (MDD), generative programming, and Microsoft's software factories [15]. Traditionally, software development is code-driven. As a result, the code and its models—if models do exist in the first place—are disconnected in the sense that there is limited traceability and synchronization [263]. In contrast, model-driven software development employs models as the primary vehicle to express software. From the models a code generator produces executable code. The code generator (or model transformer) embodies the reusable part that can be (re)applied to different models. Typically, the code generator will not generate the complete application's code, but rather fill in missing pieces in an existing code base (e.g., instantiation of an object-oriented framework). As a result, models are executable in the same sense that source code is an executable specification.

Rugaber and Stirewalt propose to apply the model-driven approach to the reverse engineering process [265]. Reverse engineering of a system would produce models that can act as formal specifications of the system under study. From these models, supported by a code generator, "another version of the original system" can be produced.

Examples of generative techniques and concepts that can be applied for MDTD are

—Traditional scanner and parser generators such as lex and yacc.

—Meta-compilation systems for generating language-based tools such as LISA and JastAdd (both based on attribute grammars).

—Meta-CASE tools, which provide facilities for specifying and generating CASE and software development environments (e.g., IPSEN, MetaEdit+, and the Generic Modeling Environment (GME)).

—Application generators such as Neighbors' Draco and Batory's GenVoca systems [264].

—Executable meta-modeling (e.g., Kermeta) and modeling frameworks (e.g., Eclipse EMF/JET).

—Meta-programming, and source transformation and rewrite systems (e.g., DMS, Stratego, ASF+SDF, Rascal, and TXL).

—Fourth-generation languages (4GLs) such as (relational) query languages and report generators.

—Generative programming techniques such as C++ template metaprogramming and aspect-oriented programming [238].

In practice, there is an overlap between the capabilities of the above techniques and the techniques are not clearly delineated. What they have in common is a domain-specific (modeling) language. These languages allow to specify the solution using problem domain concepts. They are often declarative (allowing the user to express *what* is to be done rather than *how* it is to be done) and based on different underlying paradigms or concepts (e.g., grammars, algebraic specifications, and first-order logic).

While CBTD is widely applied, to our knowledge there are only few examples of MDTD in the reverse engineering domain. Favre emphasizes the importance that meta-models have in software construction [266]. Each software artifact (e.g., source code, test case, bug tracking entry, database record, and XML document) is, in fact, a model that conforms to a meta-model. The meta-model is not necessarily explicit, but may be implicitly encoded in the model's operation.[17] Metaware is a software that operates at the level of meta-models (e.g., compilers, IDEs, testing frameworks, databases management, and XSLT). Explicit—or reified—meta-models that are easily processable by metaware are an important step toward MDTD. Favre has implemented the G^{SEE}, which is a meta-model-driven tool that interprets a meta-model specification and customizes the tool accordingly [123, 136].

The Moose reverse engineering environment is based on an executable meta-model (EMOF 2.0 compliant) implemented in Smalltalk [267, 268]. This approach allows it to not only specify the meta-model, but also to attach behavior to it in the form of Smalltalk code. The implementors of Moose justify this approach as

[17] To give an example, an XML file may have an explicit meta-model in the form of a DTD or XML Schema. If an explicit form is missing the schema may be implicitly encoded within the tools that read the XML file and the tools' expectation of well-formed input.

follows: "we felt the need to meta-describe our environment to enable us to be more efficient building new tools for our reengineering research. Using meta-modeling was just a means to introduce more flexibility and extensibility in our tools" [267]. The meta-model is implemented in a dedicated framework, called Fame, and leverages Smalltalk's reflection capabilites, class extension, and pragmas. With Fame, functionality such as serialization and UI behavior can be realized in a generic manner by operating on the meta-level. Note that Fame is based on an interpretative approach (i.e., no code is generated from the model).

In contrast, the developers of the REforDI tool use "reusable frameworks, formal specifications, and generators to reduce the implementation effort" of their tools [269]. They explicitly motivate this decision with the observation that "hand-coding of re-engineering tools is a painstaking business." The REforDI tool uses TXL and PROGRES specifications for design and code transformation as well as graphical, interactive functionality of the tool. According to the authors, the tool requires less than 1000 lines of handwritten C code. However, they caution that "the concept of graph grammars and the PROGRES language might be a bit difficult to learn for newcomers" [269].

Another example of MDTD is model-driven visualization (MDV) [270, 271]. MDV's vision is to leverage model-driven design for creating software visualization so that "researchers and tool designers will be able to spend more time designing and evaluating their tools and less time building them" [270]. MDV's reference architecture is depicted in Fig. 10. Both software and visualizations conform to a software meta-model and various visualization meta-models, respectively. The visualization meta-models cover domains such as graph-based, tree-based, and chart-based visualizations. At the meta-model level, transformations are applied that map from the software meta-model to visualization meta-models. A transformation encodes the abstract and visualize/synthesize activities of the reverse engineering process (Section 1.1).

The authors have instantiated the reference architecture in an Eclipse-based framework. The source meta-model adheres to the Dagstuhl Middle Model, the visualization meta-models are described with Emfatic/EMF, and the transformations are written with the Atlas Transformation Language (ATL). Figure 11 gives a toy example that shows how entities from a higher level source code meta-model (with File, Function, and Call entities) could be mapped to a graph-based visualizer meta-model (with Graph, Node, and Edge entities, respectively) [266].

3.4 Discussion

Researchers are pursing CBTD and MDTD because they are hoping to become more productive in tool development. Increased productivity in tool building frees resources for other research activities and leads to a faster innovation cycle.

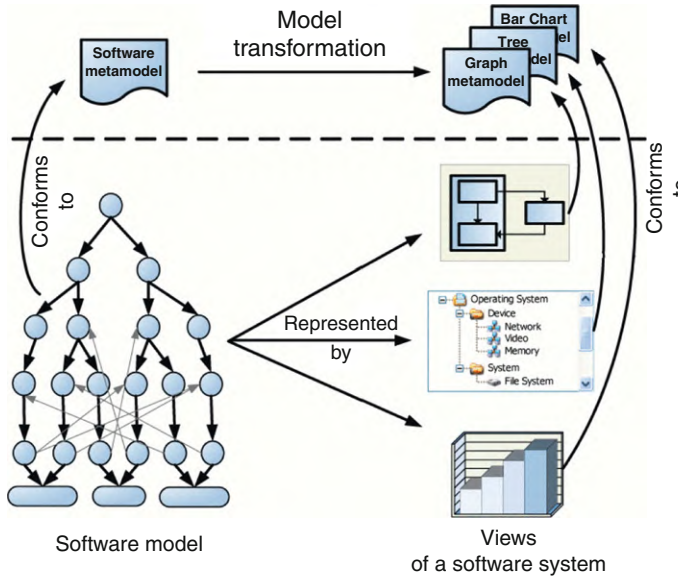

FIG. 10. MDV reference architecture [270].

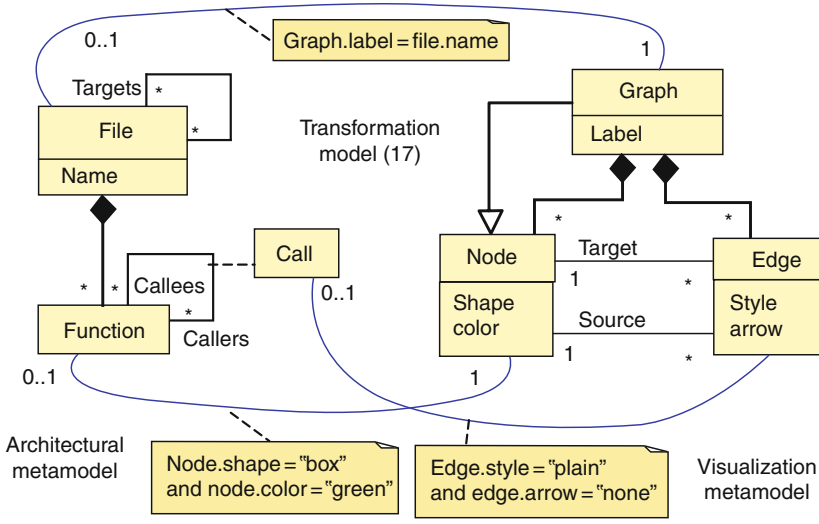

FIG. 11. Mappings between source and visualization meta-models [266].

These approaches to tool building are based on existing techniques—CBTD relates to CBD and MDTD relates to MDD—that are also used for the construction of software in general. This is desirable because they can leverage existing methodologies, technologies, tools, and experiences. However, since the academic tool-building domain has distinctive characteristics it appears that the existing, generic approaches should not be applied blindly. In order to establish best practices for tool building researchers have to elevate this lens to a first-order research topic. A yardstick for success of the tool construction lens is the generation of results that make tool building in academia more predictable and effective.

This means, for example, establishing workshops about tool building and publishing issues about tool building. In fact, there are encouraging signs in that direction. For example, Coppit and Sullivan have introduced an approach to tool construction that they call package-oriented programming (POP). In a sense, POP is an instantiation of CBTD that focuses on "the use of multiple, architecturally compatible, mass-market packages as large components" [272]. The use of multiple components is motivated by the observation that many tools require functionality that needs to be drawn from several independent domains such as text editing (e.g., provided by Word) and graph editing (e.g., provided by Visio). POP proposes to use components that are architecturally compatible to simplify integration and to minimize architectural mismatch.

Furthermore, reverse engineering researchers of successful tools have published about their experiences. For example, Lanza describes his experiences with the CodeCrawler software visualizer [273], discussing CodeCrawler's architecture (composed of three subsystems: core, meta-model, and visualization engine), the visualization engine (realized by extending the HotDraw framework), and desirable interactive mechanisms for usability. Furthermore, he distills lessons learned for all of the discussed issues. He observes that "to our knowledge there is no explicit work about the implementation and architecture of reverse engineering tools, and more specifically about software visualization tools." Guéhéneuc describes his use of design patterns and Java language idioms when constructing the Ptidej tool suite [274]. In addition, it would be highly desirable to identify lessons learned that generalize over individual tool building experiences. Currently, there are few examples of researchers that have published such lessons learned (e.g., [169, 275, 276]).

Research is also needed in technological issues. For example, an important topic is effective tool integration. Wuyts and Ducasse describe a tool integration framework based on Smalltalk and how it is used in the StarBrowser [277]. Another example is the push to establish a discipline for software that depends on grammars (so-called grammarware) by Klint et al. [278]. It is certainly the case that reverse

engineering tools are grammarware. The development of parsers for reverse engineering is characterized by *ad hoc* approaches rather than engineering. Klint et al. provide a first step toward grammarware engineering by describing a number of principles that should be followed, a life cycle for grammarware, and a list of research challenges.

Process is a neglected area with respect to tool building. It seems that few academic tool building efforts make use of an explicit process. This is hard to justify, because any software should be constructed based on a process. However, a process has to take care not to stifle unnecessarily the creative elements in research. As a first step, we have proposed a dedicated process framework for tool building in academia [279]. This framework is lightweight and makes allowance for the diversity of academic research projects (e.g., tool requirements, degree of technical uncertainty, complexity and size of the problem, and number and expertise of the development team).

Besides individual efforts, there needs to be a recognition of the tool building lens by the research community. One forum for researchers to meet and discuss tool building issues is the *International Workshop on Academic Software Development Tools and Techniques* (WASDeTT) [280, 281]. WASDeTT was held twice in 2008 and addresses topics such as

—Language-independent tools: How can we build tools that work across multiple languages?

—Tool building in an industrial context: How to build tools that get adopted by industry?

—Data interoperability among tools: How to exchange data between tools and how to process this data?

—Maturation of tools: How to grow a tool from an early prototype into a mature tool or framework?

—Tool building methodology: How—and to what degree—can we adopt established software engineering techniques for building research tools?

—Tool building in teams: How to build tools in larger—and possibly distributed—teams?

—Tool implementation language: How does the choice of a programming language impact the building of a tool, its usability, and the context in which the tool can be applied?

Some instances of WASDeTT are coupled with special issues on *Experimental Software and Toolkits* (EST) [282]. In fact, the organizers of the first WASDeTT say that "one important goal of this workshop series is to enable researchers to publish about their tools so that they can get scientific credit for their tool building efforts" [280].

4. Tool Evaluation Lens

> "Evaluate the effectiveness of reverse engineering tools and techniques through empirical studies."
>
> Wong [62, Requirement 23]

It is not sufficient to build a reverse engineering tool for its own sake. The effectiveness of the tool has to be evaluated as well in some form. This can be accomplished with empirical research, which is a field that studies real-world phenomenon. In the context of tools this means to study how tools are used by certain subjects and how effective the tools are in accomplishing certain tasks on certain kinds of systems. In software engineering, the emphasis has been on development of new technologies rather than on evaluation and comparison of the effects of these technologies [283]. Similarly, reverse engineering tends to focus on the building of new tools, rather than evaluating these tools. However, both tool building and tool evaluation are need to establish reverse engineering as a science.

A tool evaluation has the following components (or treatment factors) [284]: (1) the tool under study, (2) a subject system that is applied on the tool, (3) tasks to be performed on the subject system with the tool, and (4) the users that operate the tool. Generally speaking, an empirical study involves the following steps [283]: (1) formulating a research question, (2) designing a study, (3) gathering data, and (4) analyzing and interpreting the data. Empirical studies of tools can be grouped into the following approaches:

Case studies: A case study investigates the tool within a real-life context.[18] As a result, a case study often has little control over the evaluation setting. Also, the tasks that are performed with the tool are typically not well described because reverse engineering tools are typically used in an exploratory style and on an as-needed bases during software development.

According to Sjøberg et al., "case studies are particularly important for the industrial evaluation of software engineering methods and tools" [283]. Case studies are perhaps the most popular approach to evaluate reverse engineering tools. However, most case studies are rather weak in the sense that the tool is applied to a smaller subject system and that the users of the tool are its developers. Few case studies are conducted within an industrial context. In the ideal case, the "tool under

[18] We use the term case study rather loosely guided by what researches in reverse engineering consider to be a case study in their publications. Many of these "case studies" may not actually conform to a more rigorous definition. Sjøberg et al. report on a literature study where they found that 58% of case study papers did not meet their definition of evaluative case study [284].

investigation is tried out on a real project using the standard project development procedures of the evaluating organization" [285].

Experiments: Experiments investigate a tool within a controlled environment with the goal to obtain certain measurements to test a hypothesis or theory. In an experiment, users of the tool can be selected and assigned to groups based on their characteristics. The tasks that are performed with the tool tend to be small, but relatively well defined. Since experiments use controlled treatment factors they are replicable and exhibit little bias.

There are very little experiments that evaluate reverse engineering tools. Storey et al. have conducted a user experiment to evaluate the effectiveness of the user interfaces of three different tools: Rigi (which uses a multiple windows approach), SHriMP views (single window), and Unix standard tools (grep and vi) [102]. The experiment measured correctness of performing a number of reverse engineering tasks and the time to complete the tasks. This experiment showed that Unix standard tools were the least effective.

Panas and Staron have conducted an experiment involving CBTD in which they used *ad hoc* composition and a framework-based approach (VizzAnalyzer) to build a reverse engineering tool [286]. They use the Goal-Question-Metric approach to answer questions about the effectiveness of tool construction and the tool quality. The experiment showed that framework customization is superior to *ad hoc* composition.

Benchmarking: Benchmarking is often associated with performance comparisons such as the well-known SPEC and TPC benchmarks. However, benchmarking also has a broader meaning that covers evaluation approaches that use a well-defined task sample associated with a performance measure so that a comparison among alternatives (e.g., different tools) is possible.

Benchmarking has characteristics from both case studies and experiments. They have qualitative and quantitative elements [284]. Benchmarking allows a direct comparison of results and has built-in replication. On the other hand, there is little control over the users of the tools and how they apply it. Consequently, benchmarks are preferable "if the tool undertakes automatic transformation of data with little or no interaction with the tool user (e.g., a speech synthesizer, or a compiler)" [285].

Sim has developed the CppETS benchmark to measure the performance of C++ fact extractors [77]. The task sample is structured into accuracy and robustness tests. Accuracy checks whether an extractor is able to produce correct facts for preprocessing directives and C++ entities such as variables, functions, and exceptions. Robustness tests issues such as missing header files, C++ dialects, and embedded languages. The benchmark was applied to four extractors (Ccia, cppx, Rigi's C++ parser, and TkSee/SN). Figure 12 shows the scores of the extractors for both kinds of tests. A drawback of CppETS is that scoring involves a significant amount of manual

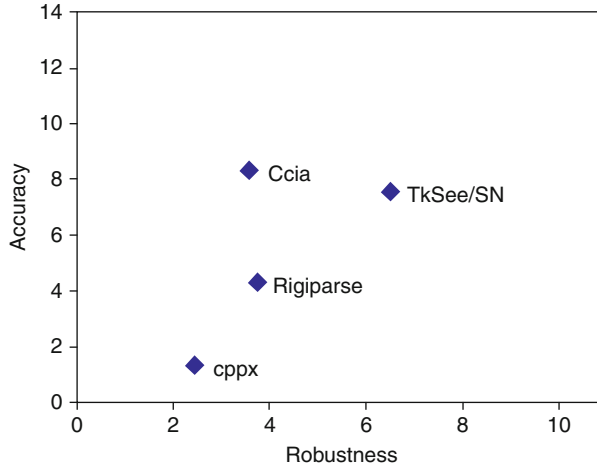

FIG. 12. Score of C++ fact extractors in the CppETS benchmark [77].

labor. In the best case, a benchmark is automated so that it can be easily rerun or applied to a different tool.

Perhaps as a result of Sim's efforts [287], the reverse engineering community has proposed a number of benchmarks (e.g., for software evolution [288], clone detection [289, 290], design pattern detection [291], and recommender systems [292]). Bellon et al. have evaluated clone detectors with the help of a benchmark [290]. All necessary information is made available so that the benchmark can be replicated. They hope that "benchmark evaluation becomes a standard procedure for every new clone detector" [290].

Feature analysis: A feature analysis is based on a number of criteria that users have for a particular reverse engineering activity and mapping those criteria to features that a tool should have in order to support the activity [285]. The criteria and features could be based on personal opinion or synthesized by means such as questionnaires or (systematic) literature reviews. Often the first step is omitted or not explicitly identified. As a result, a feature analysis is then focused on how well a tool meets a given number of desired features.

Researchers often use feature analysis to evaluate a given tool. For example, the developers of Sextant evaluate their tool with five functional requirements. Feature analysis is also popular to compare different tools among each other. For example, Bellay and Gall have compared the capabilities of four reverse engineering tools (i.e., Refine/C, Imagix 4D, Rigi, and SNIFF+) in terms of their general capabilities,

Assessment criteria	Refine/C	Imagix 4D	Rigi	Sniff+
Analysis	++	++	0	0
Parsable source languages	C (ANSI), C (K&R)	C (ANSI), C++	C (ANSI), C++, COBOL, PL/AS, Latex	C (ANSI), C++, IDL
Other importable sources	/	gdb and tcov results	/	/
Incremental parsing	–	+	–	+
Reparse	–	+	+	+
Fault tolerant parser	–	–	–	+
Quality of parse error statements	0	+	+	–
Parse abortable	+	–	+	–
Parse errors viewable during parsing	+	–	+	–
Parse stopped at error or continued	Stopped	Continued	Continued	Continued
Point and click movement from parse results to source code	+	–	–	–
Parsing results	++	++	0	–
Parsing speed	0	0	+	+
Define and undefine	++	++	0	0
Preprocessor command configurable	+	–	–	+
Support for additional compiler switches	+	+	+	+
Project definition	File	File, directory, makefile	File	Directory

FIG. 13. Feature analysis of the analyzer capabilites of four tools [73].

analysis, representation, and editing/browsing [73]. Figure 13 shows the feature list that compares the tools' analysis capabilities.

There are also a number of other tool comparisons and comparative frameworks as already described in Section 2. For example, Guéhéneuc et al. introduce a comparison framework for design recovery and apply it to two tools [78]. The framework addresses eight concerns: context, intent, users, input, technique, output, implementation, and tool. Some tool comparisons incorporate benchmarks, but feature analysis is mostly qualitative.

Structured tool demonstration and other challenges: A structured tool demonstration is "a hybrid evaluation technique that combines elements from experiments, case studies, and technology demonstrations" [293]. Such a demonstration was held as part of a CASCON workshop where different teams had to perform certain reverse engineering tasks with their tool on the same system in a live setting. Thus, a demonstration covers the entire reverse engineering experience in a day. Each tool is assigned an impartial observer who records how the tools are used to solve the tasks and who acts as "apprentice," trying to become proficient in the tool. The organizers of the event emphasize that the purpose of such a demonstration is not to establish a ranking but rather to give the participants insights into their own tools. They pose the thesis that

> "a structured demo provides a lot of insight for tool designers into their own tools and allows them to directly compare their tool capabilities with other tools and learn about future tool extensions"
>
> Sim et al. [151].

In a follow-up study, two additional tools (GUPRO and Bauhaus) performed the same tasks on the same subject system, but not live [151].

Other more informal tool challenges have been conducted as well. These challenges have less control over the treatment factors which means that they rely on personal interpretation, and do not allow comparability or reproducibility. The perhaps first event of this kind was a call to test different tools on an industrial legacy system [294]. The organizer expected 20–30 participants and results were presented in a session at CSMR 1998. In another event, eight tools did participate in the analysis of the SORTIE legacy system (30,000 lines of Borland C++) [295]. The teams had the task to rearchitect the system and to submit a report. Results were presented at WCRE 2001.

The working conference on *Mining Software Repositories* (MSR) has established the *Mining Challenge* since 2006. The challenge focuses on a particular software system (e.g., Eclipse) but has no fixed task sample nor does it prescribe a task. Similarly, VISSOFT 2007 did feature a Tool Demo Challenge for visualization tools to perform a number of suggested tasks (e.g., architecture, source code, or evolution analysis) on Azureus or GCC. Three tools did participate (CGA Call Graph Analyzer, Rigi, and VERSO) and submitted short reports.

Empirical research plays a key role for two interrelated activities: the evaluation of an existing tool and the generation of theories that inform the construction of tools. We call the first activity *evaluation-driven tool building* and the latter one *theory-grounded tool building*, discussing them in Sections 4.1 and 4.2, respectively.

4.1 Evaluation-Driven Tool Building

> "In general, the testing of the effectiveness of many tools has been seriously lacking. The value of many research ideas have not been adequately substantiated through empirical studies."
>
> Storey et al. [24]

We define the term evaluation-driven tool building to emphasize that tool evaluation should be an integral part of a tool building effort. Evaluation should be considered not only as an afterthought but also during the whole tool building project.

Sensalire et al. describe a tool evaluation cycle that consists of four steps [296] (Fig. 14). This cycle emphasizes that tool building is an iterative activity that

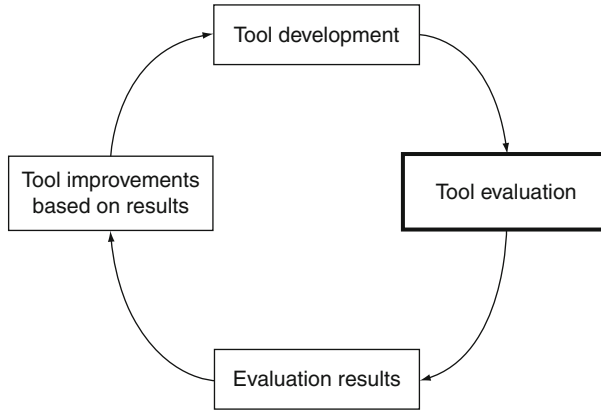

FIG. 14. Tool building and tool evaluation cycle [296].

receives feedback by conducting evaluations. In the reverse engineering community, there is increasing realization that tool evaluations are needed to advance the field (e.g., [156, 203, 296]). Concerns that proper tool evaluations are needed have been voiced in the community at least for a decade (e.g., [24]).

However, tool evaluation is not yet a standard practice. In order to establish tool evaluations more firmly, there needs to be a consensus in the community what constitute a "proper" evaluation. Typical examples of tool evaluations that can be found in the literature are

—Personal impression of the tool developers.
—Anecdotes of tool usage reported back to the tool developers.
—Checking the tool's features against a list of requirements (Section 2).
—Observational user studies.

Evaluations of tools involving users are mostly conducted with a smaller number of subjects. Furthermore, these subjects are often from a purely academic background. Less than 10% of the empirical studies in software engineering use professional software developers [299]. For reverse engineering, this number is probably even lower, leading to the question—perhaps voiced in exasperation—of "by the way, did anyone study any real programmers?" [27].

Many user studies are controlled experiments (i.e., they are conducted in an artificial environment, *in vitro*). In addition, more studies that evaluate a tool's adoption and use in industry (i.e., natural work environment, *in vivo*) would be highly desirable. Such studies are less controlled by their nature, pose logistical challenges, and "may result in vast amounts of data that is difficult to analyze"

[203]. Controlled experiments with their quantitative nature should be complemented with qualitative studies that ideally observe industrial programmers [300].

Another concern is that most user studies have a small sample size. Huang and Tilley observe that most software engineering studies have this problem: "In most software engineering studies, researchers are often pleased if they can attract a dozen students to participate in their experiment" [202]. Sjøberg et al. report in a survey that on average studies have 49 subjects (with a wide range from 4 to 266) [299]. Studies with a small sample size suffer from the problem that their statistical significance is questionable. There are large studies in software engineering such as an evaluation of pair programming that involved 295 professional software developers [301]. Unfortunately, to our knowledge there are no studies for reverse engineering that are even remotely comparable in terms of size and effort.

Empirical studies can be distinguished by their *distance from human contact* [27]: first-degree contact exhibits a direct involvement of study participants, second-degree contact exhibits indirect involvement of participants by collecting data about them, and third-degree contact relies on theories that have been informed by lower degrees of contact. User studies are first-degree with respect to the human contact. While first-degree of contact has many benefits, it also poses potential problems. Storey cautions that "observations can also be disruptive and could be subject to the Hawthorne effect (e.g., a programmer may change her behavior because she is observed)" [203]. If students are involved there is the added problem that they "will often feel pressured to participate in their professor's research even if they are assured that participation is voluntary" [302]. Also, Berry and Tichy state that "students in particular have a strong desire to please their instructors" [303]. These problems are mitigated by evaluation approaches that are based on second-degree (e.g., instrumentation of tools to collect usage information [304]) and third-degree (e.g., application of feature analysis to evaluate the tool).

Importantly, evaluation is not necessarily a "big-bang" activity that happens after the tool has been "finished." Instead, evaluation issues should be considered as early as possible in the development effort. For example, user interface evaluations can be already conducted before functionality is implemented with the help of paper-based prototypes. When designing the tool it can be useful to consider already how the tool can be instrumented to record user interactions during an experiment. Evaluations can be designed based on the tool's expected research contributions. Such evaluations should be planned during tool development since experiments involving users can take a long time to plan and to get approved by ethics committees. Tool prototypes can be used to evaluate certain aspects of the tool in a lightweight, informal manner. For example, Google uses agile techniques such as "guerilla usability testing (e.g., limited numbers of users hijacked from the Google cafeteria at short notice)" [305]. A promising approach that enables the evaluation

of tools while they are being developed is a process based on (evolutionary) prototyping (e.g., SEI's Evolutionary Development method [64]).

4.2 Theory-Grounded Tool Building

"The best theory is inspired by practice and the best practice is inspired by theory."
Knuth [306]

The purpose of a theory is to explain a set of empirical observations. In tool building, a theory can be used to guide an empirical study and to explain its outcome. Also, a theory can be useful to drive tool requirements. Storey argues that theories are needed as a solid foundation for research [203]:

> "Irrespective of the evaluation technique used, theoretical underpinnings will benefit the evaluations as the results will be easier to interpret. Although our long term goal may be to build better tools, we need to understand *why* they are better than other approaches."

In the following, we use the term theory rather loosely—incorporating (working) hypotheses, scientific laws, recurring experiences, lessons learned, etc.—including it to mean a proposed explanation for a phenomenon that is able to improve upon tool building.

An experiment without an underlying theory is mostly only able to establish that the tool has a characteristic based on some measure (e.g., superiority of a tool in terms of maintenance speed). Thus, the experiment is able to support a claim, but lacks a reasoning why this is so. Theories provide a framework for experiments by guiding the setup, focusing the observations, and providing reasoning for the outcomes [156]. There is a large number of potential theories from various scientific fields that can be applied to reason about tools. For example, program comprehension can be investigated at least with cognitive, psychological, and socio-cultural theories [300]. Exton proposes to utilize constructivist learning theories for tool building [307]. He notes, for instance, that "constructivist learning environments provide multiple representations of reality." An implication for program comprehension tools is that they should have "the ability to present the same information with the same level of abstraction but with a different emphasis."

While cognitive theories are suitable to study individuals, theories from social sciences are needed if tool is used in a collaborative work style [16]. In this context, O'Brien et al. recommend to incorporate socio-cultural theory, believing that "socio-cultural psychologists would consider that research methods focusing solely on the individual in a purely experimental environment are deficient and incapable of gaining a true insight into how understanding occurs. In fact, most empirical work

carried out to date does not take into consideration external validity in terms of the programmers' environment, the source code used in these studies, or the tasks required of participants" [300]. Theories about adoption and technology transfer then can be applied to reason about tool adoption in an organizational setting (Section 2.5). This has been advocated by the Adoption-Centric Software Engineering project, which treats tool adoption as a first-class goal.

Instead of applying theories with little or no modification, general theories can be leveraged as a foundation to formulate dedicated theories for tool building and the reverse engineering domain. There are a number of different program understanding strategies that derive from cognitive theories (i.e., bottom-up, top-down, knowledge-based, opportunistic vs. systematic, and the integrated model) [203]. For example, Solloway and Ehrlich's theory of top-down understanding "borrows" from two sources: text comprehension and problem solving theories [308]. Program comprehension theories can be used to come up with tool requirements. In fact, Storey observes that "many of the researchers that developed the traditional cognitive theories for program comprehension discuss the implications of the developed theories on tool design" [203]. However, she also concludes that "in many cases, the connection to tools and how they could be improved or evaluated according to the theories could be stronger."

Formation of theories is often grounded in (empirical) observations rather than controlled experiments; the latter are more suitable for testing theories. Theories for program comprehension are often based on observation of programmers and their work practices. Examples of observational techniques are work diaries, think-aloud, shadowing, and participant observation via joining the team [309]. Lethbridge and Singer have developed a method called Work Analysis with Synchronized Shadowing (WASS) to study and record the work of software engineers [27]. Synchronized shadowing uses two note-takers during the observation that place different emphasis on the kinds of data that they are collecting. The records are later merged and encoded in use case maps, which show the flow of tasks including tools such as grep and emacs, contexts such as documents and other people, and actions. Based on the use case maps, work patterns can be synthesized whose interpretation can lead to theories and tool requirements. Lethbridge and Singer used WASS to study eight industrial software engineers and were able to derive a number of requirements for software exploration tools. For example, they found that a "tool should have a simple command to automatically locate occurrences of whatever is in the copy buffer" [27]. This is based on their observation that users frequently used the copy buffer as the argument when performing a search with the editor.

Mayrhauser and Vans use their integrated model of program comprehension to study industrial programmers with think-aloud doing maintenance tasks. Based on their observations, they map maintenance tasks to information needs and tool

capabilities [310]. For example, they found that programmers want to come back to a code segment that they looked at previously. Thus, they have the information need of "browsed locations" which a reverse engineering tool can satisfy by offering a "history of browsed locations." More recently, Ko et al. have conducted an experiment with students performing maintenance tasks in Eclipse [311]. They say that "the central goal of our study was to elicit design requirements for tools to help with maintenance tasks" and consequently propose six such requirements and how a tool could realize them. For example, their study suggests that part of a task entails "to collect a working set of task-relevant code fragments." In Eclipse, part of the working set is represented by the open file tabs and the state of the package explorer. As a consequence, switching of tasks results in a loss of the working set that needs to be manually recovered. They propose that a tool should "provide a working set interface that supports the quick addition and removal of task-relevant code fragments." In a sense, these findings refine Mayrhauser and Vans' observation that a history of browsed code locations is important during maintenance.

For tool building, theories constitute existing knowledge that can be used to build better tools. In this context, it is important to "package" theories in such a form that they can be more easily applied by researchers that are primarily interested in tool building as opposed to advancing and developing theories. Walenstein has done extensive research in how to apply cognitive theories to tool building, distilling his findings in a cognitive modeling framework (HASTI) that can be used to reason about cognitive support [312]. He explicitly states that "HASTI is tailored specifically to the needs of application-oriented researchers. They need abstractions and simplifications such that the important issues of cognitive support can be efficiently raised and addressed. They also need prebuilt models that can be rapidly and widely applied to yield insight" [156]. Examples of other cognitive theories that can be applied by tool builders are Green and Petre's cognitive dimensions framework [178], Storey's cognitive design elements for software exploration [163, 164], and Murray's cognitive patterns (or microtheories) for program comprehension [313, 314]. Storey has applied her design elements to guide and assess the SHriMP visualization tool; similarly, Farah has applied Murray's patterns for her Temporal Model Explorer tool [315]. The patterns enabled the generation of a number of (candidate) tool features.

A promising approach for generating theories for program comprehension is grounded theory, which is commonly applied in areas such as cognitive science and psychology. With grounded theory, new theories are synthesized bottom-up from experimental data rather than being informed—and constrained—by existing theories. Additional data leads to a refinement of the theory so that it continues to support all available data. Researchers in the reverse engineering domain have both advocated (e.g., [16]) and applied (e.g., [27, 316]) grounded theory. However, it is

crucial to not abuse grounded theory as a fig leaf for unsystematic theory generation. Based on their experiences, Adolph et al. caution that "like many other researchers who have claimed to follow grounded theory methods and even produce a grounded theory, many of us have only borrow[ed] a few grounded theory practices and have not followed grounded theory as a comprehensive method" [317].

4.3 Discussion

> "Scientific rivalry between experimenters and between tool builders can thereby lead to an exponential growth in the capabilities of the tools and their fitness to purpose."
>
> Hoare [318]

The key elements of empirical research in the reverse engineering domain—theories, experiments, and tools—are mutually dependent on each other. This is illustrated in Fig. 15. Theories can guide tool experiments (e.g., Walenstein's HASTI [312]) and to a lesser degree provide requirements for tool building (e.g., Mayrhauser and Vans [310]). Experiments can be used for tool evaluation and to confirm or refute theories. Same as theories, experiments can provide requirements for tool building (e.g., Lethbridge and Singer [27]).

An approach to tool building that is evaluation-driven and theory-grounded has to follow a suitable process that incorporates theories, experiments, and tool building. An example of such a process that incorporates these elements is described by Storey [163]. The process consists of "several iterative phases of design, development, and evaluation" and has been used on the SHriMP tool. Figure 16 depicts a rendering of the process iterations with SHriMP as the subject tool. There is an "iterative cycle of design and test" that aims at improving a tool [164]. The (initial) design of the tool is guided by Storey's cognitive design elements framework, which is a theory that provides a catalog of issues that should be addressed by software exploration tools. Evaluation of the tool's design and implementation can be accomplished by empirical studies based on observation of tool users. This cycle is

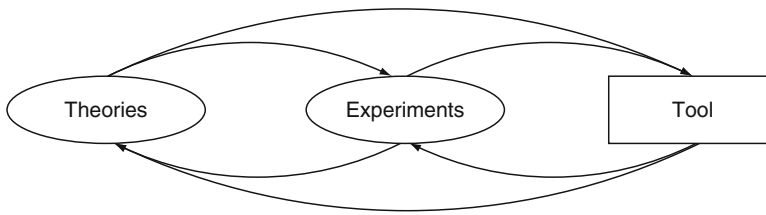

FIG. 15. Cycles of tool construction, experimentation, and theory creation.

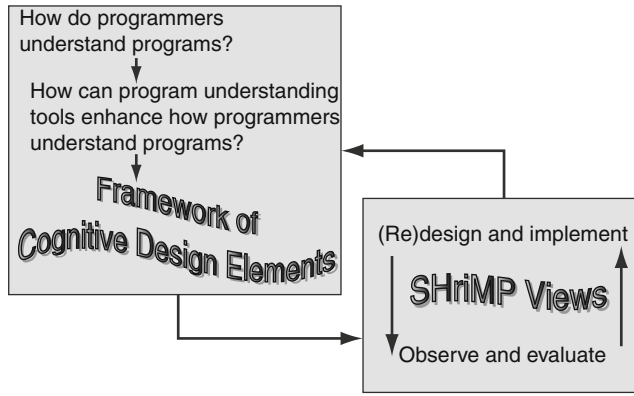

FIG. 16. Storey's process consisting of two feedback cycles [163].

embedded in a larger iterative cycle for improving the underlying theory. When adopting Storey's process, the cognitive design elements framework could be replaced with a different theory.

Compared to the other two lenses, tool evaluation is still in its infancy. However, the evaluation of tools has made rapid advances within the last few years, which is a source for optimism that some form of empirical evaluation will become an expected component when reporting a research result. However, empirical studies are costly and thus should be conducted strategically (e.g., based on cost-benefit concerns). For example, the goal is not to maximize the quality, but to find the right level of quality [283]. Having a large number of lower quality studies that are only conducted as a token effort to get a tool or method published may turn out to be detrimental to the long-term impact that empirical studies can have in synthesizing broadly accepted theories. In fact, if empirical studies are not following a rigorous procedure, the resulting data "may fail to support even a true hypothesis, or, conversely, false hypotheses may be believed to be true" [319]. It is now expected that an empirical study discusses threads to validity. However, instead of repeating (boilerplate) discussions for each evaluation, the effort should go toward reducing the threads. Also, not everything needs to be empirically investigated (e.g., it may be sufficient to rely on qualified opinions or long-term anecdotal evidence). For example, Parnas points out that the effectiveness of Kirchhoff's circuit law has not been empirically evaluated—its effectiveness is simply accepted by electrical engineers [320].

Key concerns for empirical research in the reverse engineering domain are the development of agreed-up standards that make it easier to conduct, compare, and reason about empirical studies. In his WCRE 2006 keynote, Briand makes the point

that "it is important that the reverse engineering research community strives to provide guidelines and develop specific empirical procedures and benchmarks that would eventually converge to become standard practices" [321]. An example of work in this direction is the proposal of a framework for defining and performing experiments for program comprehension, and of collaborations among researchers where each participant contributes to a study in a different way [298, 322]. Researchers have also proposed a common coding scheme for observational studies [323]. Reverse engineering can benefit from software engineering, which is facing similar challenges [324]. Zelkowitz and other researchers advocate the sharing of information related to empirical studies in software engineering among research groups [325]. Examples of information that should be shared are experimental results in the form of quantitative data; artifacts that support an experiment such as code, design documents, and test plans; tools for data collection; and procedures and guidelines such as reporting forms.

A straightforward approach to advance empirical studies is to work toward a benchmark, or at least a common set of subject systems. A benchmark is relatively easily applied and promises replication. Without a standard set of subject systems and evaluation principles encoded in a benchmark, there is the risk that tool evaluations are biased by applying a new tool or technique to favorable software systems. In addition, defining a benchmark can have a community-building effect, advancing the discipline: "Throughout the benchmarking process, there is greater communication and collaboration among different researchers leading to a stronger consensus on the community's research goals" [287]. Communication of lessons learned are another example that can help to advance empirical research. Sensalire et al. describe lessons learned based on five different user studies with software visualization tools [296]. Adolph et al. discuss lessons learned in applying grounded theory in the software engineering domain [317].

A key problem of empirical research is lack of knowledge and interest. Reverse engineering researchers that are willing to advance the state of empirical research (e.g., by applying grounded theory) are rather ill equipped because their background has not prepared them to apply social science research methodology [317]. Huang and Tilley provide an example that affirms this point [202]:

> "We are novice empiricists who are interested in using empirical studies as a measurement instrument; we are not particularly interested in becoming experts concerning empirical methods *per se*. As novices, we struggled with learning how to properly execute an empirical study, while at the same time remaining focused on the underlying problem that we were interested in exploring."

Also, researchers that have chosen an engineering discipline are often not particularly keen on becoming experts in empirical research. One avenue to raise the

quality of empirical studies and the construction and application of theories is that the reverse engineering community works toward establishing a research environment that encourages researchers to specialize in empirical research involving tools. Such a research environment would need to give adequate scientific credit for empirical work (e.g., synthesizing of tool requirements and replication of tool evaluations) on a par with nonempirical work. This would lead to a mutual fertilization of researchers that focus on tool building and tool evaluations. Walenstein describes a grand vision of theory-based research in which multidisciplinary collaboration would lead to theories that serve as a foundation for software engineering research. In this vision, "software engineering researchers will be portrayed as consumers of theoretical advances from other disciplines, the focus of the software engineering thread of research will be set on applications of theories derived elsewhere" [156, p. 56].

5. Conclusions

In this chapter, we have addressed the issue of academic tool building in the reverse engineering domain. We have explored this issue with the help of three lenses: tool requirements, tool construction, and tool evaluation. Each lens represents an opportunity to reflect upon the current research practice and how to improve upon it.

Each lens has the potential to positively influence other lenses, which in the best case may lead to a positive feedback cycle. Articulating tool requirements represents an opportunity to make research goals more explicit. For example, what are the desirable quality attributes that the tool has to meet? It is expected that this set of requirements will be changed and refined as research advances and the tool gets constructed. Thus, tool requirements and tool construction are mutually influencing each other. Initial tool requirements can serve as early guidance for tool construction. For example, quality attributes can inform tool architecture. Conversely, experiences gained by building and test-driving a tool prototype can refine the tool's requirements.

The tool's quality attributes also depends on the evaluation. For example, what are the minimum requirements so that the tool can be used for the envisioned tool evaluation? If the tool is evaluated by the tool developers themselves (e.g., benchmarking its scalability) then usability and adoptability are less of a concern. Conversely, if the tool is evaluated by a user study then a certain level of usability has to be met. Furthermore, the tool has to scale up to accommodate at least the evaluation environment, which may be a subject system with millions of lines of

code in an industrial setting. A key motivation of empirical studies is to show that a new tool or technique provides a measurable benefit compared to the state-of-the-practice. Finkelstein and Kramer point out that researchers "cannot expect industry to make very large big-bang changes to processes, methods and tools, at any rate without substantial evidence of the value derived from those changes" [106]. Thus, convincing empirical studies can support tool adoption by industry. While there is a considerable attention on process in software engineering, the process of developing tools in academia is still neglected. Oivo proposes to make the development work in a research project the subject of empirical study, treating it as an experiment or quasi-experiment [326]. This approach provides a more realistic environment for experimentation compared to small-scale user studies involving students. It has the potential to strengthen empirical results and to introduce more rigor into academic tool building.

Besides the discussed lenses, reverse engineering research is experiencing a number of new challenges due to novel perspectives on how to reverse engineer systems and due to the shifting nature of the systems themselves. These challenges reflect back on the lenses in terms of new emerging tool requirements, new tool architectures, and new avenues for empirical evaluations.

Research in software evolution has led to the realization that there is no strict separation between the development and maintenance phase of a software system [10]. Similarly, reverse engineering activities are not removed from forward engineering activities—both are rather intertwined. This understanding has consequences on reverse engineering tools. First, it leads to a tighter integration of the reverse engineering tools with the software development tools, meaning in practice that reverse engineering functionality is integrated seamlessly within the developers' IDE. Second, reverse engineering functionality has to be accessible not only instantaneously but also produce results instantaneously. Thus, batch-processing of tasks that run for a longer time period are out-of-step with a highly interactive, IDE-driven working style. In a sense, each additional second that an analysis takes rapidly decreases its attractiveness for the tool user. Thus, the idea of JITC is complemented by real-time comprehension and real-time reverse engineering. Third, following the lead of IDEs to become more collaborative, reverse engineering tools are following suit. Incorporating collaborate features enables novel reverse engineering functionality, not just augmenting existing reverse engineering approaches with a collaborative touch. However, this also leads to new challenges for tool evaluation because studies now have to cope with multiple tool instances that operate in a collaborative environment. Fourth, tighter integration of reverse and forward engineering in a single environment means that reverse engineering tools have access to and can leverage data sources that were previously not available. If data about every single keystroke of the user becomes available for data

mining, powerful recommender systems that operate in real-time become feasible. Thus, we expect that reverse engineering will increasingly incorporate and leverage functionality such as can be found in recommender systems. Furthermore, these new data sources can be used in empirical studies of tool usage.

In a user study involving software visualization tools for performing corrective maintenance, the participants questioned the need for dedicated tool support, voicing their preference for the Eclipse IDE [75]. This may be explained with the fact that Eclipse is offering functionality that was previously only provided by reverse engineering tools. Either way, it shows that reverse engineering tools are now competing with vanilla IDEs. If research in reverse engineering wants to stay relevant it has to show that its tools are useful beyond the state-of-the-art of industrial software development environments. A study of doing maintenance with Eclipse showed that only 20% of the time is spent editing code besides other activities such as reading code (20%), searching for names (13%), and navigating dependencies (16%) [311]. The study also exposed the potential of new tool support that could save up to 35% of the time spent on code navigation during maintenance tasks. Thus, dedicated tools for program comprehension and reverse engineering have the potential to make a huge impact if they support the right functionality (i.e., usefulness) together with the right interactions (i.e., usability). However, the increasing complexity of the underlying reverse engineering infrastructure and the pressure to compete with polished industrial tools may necessitate a community effort to collaborate on a common infrastructure (e.g., in the form of an application framework) for doing joint reverse engineering research [327, 328]. Such an infrastructure should allow "the state-of-the-art to be readily applied" and thus to advance upon it more easily [327].

The constantly changing nature of software means that reverse engineering approaches have to adapt accordingly. Traditionally, reverse engineering is concerned with large monolithic systems that are coded in a statically typed high-level language. Over the years, established programming languages have been getting more dynamic in nature (e.g., C++'s RTTI and C#'s Reflection.Emit) and new languages already have more dynamic features (e.g., Ruby). Domains such as web applications often rely on scripting languages on the client side (e.g., JavaScript) and server side (e.g., PHP). This trend has an impact on reverse engineering tools because static analyses are becoming less precise and whole program analyses are becoming impractical [297]. Thus, dynamic analyses are playing a more prominent role in dynamic languages to complement statically derived information.

Besides programming languages, the targeted systems are changing their characteristics. Systems are becoming less monolithic, more dynamic, and more adaptive in nature. This trend is exhibited by approaches such as SOA (where services are loosely coupled and can be discovered during run-time) and self-adaptive systems (where the system reconfigures itself during run-time) [329]. These kinds of

software require analyses that monitor the system at all levels of granularity and corresponding visualizations [330]. Such systems also often have components that are black boxes in the sense that their inner workings and source code are not accessible for reverse engineering. Thus, reverse engineering techniques may have to rely primarily on analyzing the interfaces and observing the interactions of black-box components. The same characteristic holds for emerging approaches such as Software as a Service (SaaS) and cloud computing, which hide most parts of the software behind the service provider's server. While such restrictions are not entirely new and can also be encountered in COTS-based and distributed systems, the increasing impact of SOA, SaaS, clouds, and self-adaptivity may necessitate radically different reverse engineering techniques.

There is also the realization that systems do not exist in isolation and have to be studied and understood in a broader context. The concept of Systems of Systems (SoS) acknowledges that the boundary of a software system is often blurred and shifting because it communicates with other systems. Also, a SoS includes systems over which the integrator has little or no control [331]. As a consequence, the subject system that reverse engineering targets may not be readily identifiable and easily bounded if that system is in fact a SoS. This problem becomes more pronounced if the reverse engineering target broadens toward ultralarge-scale (ULS) systems [332].

Looking at the challenges ahead, research in reverse engineering promises to remain exciting, despite—or perhaps because of—its growing maturation in all three lenses. While tool building is a means to an end, it does no harm that it has a very attractive characteristic—it is fun!

References

[1] P. Samuelson, S. Scotchmer, The law and economics of reverse engineering, Yale Law J. 111 (7) (2002) 1575–1663.
[2] B.C. Behrens, R.R. Levary, Practical legal aspects of software reverse engineering, Commun. ACM 41 (2) (1998) 27–29.
[3] V. Raja, K.J. Fernandes (Eds.), Reverse Engineering: An Industrial Perspective, Springer Series in Advanced Manufacturing, Springer-Verlag, Berlin, 2008.
[4] E.J. Chikofsky, J.H. Cross II, Reverse engineering and design recovery: A taxonomy, IEEE Softw. 7 (1) (1990) 13–17.
[5] E. Eilam, Reversing: Secrets of Reverse Engineering, Wiley, New York, NY, 2005.
[6] R.S. Arnold (Ed.), Software Reengineering, IEEE Computer Society Press, 1993.
[7] A.E. Hassan, R.C. Holt, The small world of software reverse engineering, in: 11th IEEE Working Conference on Reverse Engineering (WCRE'04), 2004, pp. 278–283.
[8] A.V. Mayrhauser, A.M. Vans, Program Understanding: Models and Experiments, Advances in Computers, vol. 40, Elsevier, The Netherlands, 1995.
[9] M.W. Godfrey, D.M. German, The past, present, and future of software evolution, in: Frontiers of Software Maintenance (FoSM 2008), 2008, pp. 129–138.

[10] T. Mens, S. Demeyer (Eds.), Software Evolution, Springer-Verlag, Berlin, 2008.
[11] A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, Boston, MA, 1986.
[12] S. Diehl (Ed.), Software Visualization, Springer-Verlag, Berlin, Vol. 2269 of Lecture Notes in Computer Science, 2002.
[13] N. Fenton, M. Neil, Software metrics: Roadmap, in: Conference on the Future of Software Engineering, 2000, pp. 359–370.
[14] T. Kühne, Matters of (meta-) modeling, J. Softw. Syst. Model. 5 (4) (2006) 369–385.
[15] R. France, B. Rumpe, Model-driven development of complex software: A research roadmap, in: Future of Software Engineering (FOSE'07), 2007, pp. 37–54.
[16] M.-A. Storey, C. Bennett, R.I. Bull, D.M. German, Remixing visualization to support collaboration in software maintenance, in: Frontiers of Software Maintenance (FoSM 2008), 2008, pp. 139–148.
[17] J. Whitehead, Collaboration in software engineering: A roadmap, in: Future of Software Engineering (FOSE'07), 2007, pp. 214–225.
[18] J. Hainaut, V. Englebert, J. Henrard, J.-M. Hick, D. Roland, Requirements for information system reverse engineering support, in: 2nd IEEE Working Conference on Reverse Engineering (WCRE'95), 1995, pp. 136–145.
[19] A. Hamou-Lhadj, T.C. Lethbridge, A survey of trace exploration tools, in: Conference of The Centre for Advanced Studies On Collaborative Research (CASCON'04), 2004, pp. 42–55.
[20] B.G. Ryder, Constructing the call graph of a program, IEEE Trans. Softw. Eng. SE-5 (3) (1979) 216–226.
[21] Y. Chen, M.Y. Nishimoto, C.V. Ramamoorthy, The C information abstraction system, IEEE Trans. Softw. Eng. 16 (3) (1990) 325–334.
[22] Y.R. Chen, G.S. Fowler, E. Koutsofios, R.S. Wallach, Ciao: A graphical navigator for software and document repositories, in: 11th IEEE International Conference on Software Maintenance (ICSM'95), 1995.
[23] H.M. Kienle, H.A. Müller, The Rigi reverse engineering environment, in: 1st International Workshop on Advanced Software Development Tools and Techniques (WASDeTT1), 2008. http://scg.unibe.ch/download/wasdett/wasdett2008-paper06.pdf.
[24] M.D. Storey, K. Wong, H.A. Müller, How do program understanding tools affect how programmers understand programs, Sci. Comput. Programming 36 (2–3) (2000) 183–207.
[25] K. Erdos, H.M. Sneed, Partial comprehension of complex programs (enough to perform maintenance), in: 6th IEEE International Workshop on Program Comprehension (IWPC'98), 1998, pp. 98–105.
[26] J. Singer, T. Lethbridge, N. Vinson, N. Anquetil, An examination of software engineering work practices, in: Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'97), 1997, pp. 209–223.
[27] T. Lethbridge, J. Singer, Studies of the work practices of software engineers, in: H. Erdogmus, O. Tanir (Eds.), Advances in Software Engineering: Topics in Comprehension, Evolution, and Evaluation, Springer-Verlag, 2001, pp. 51–72. Ch. 3.
[28] R. Holt, Software architecture as a shared mental model, in: 1st ASERC Workshop on Software Architecture, 2001. http://webdocs.cs.ualberta.ca/~kenw/conf/awsa2001/papers/holt.pdf.
[29] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, C. Riva, Symphony: View-driven software architecture reconstruction, in: 4th IEEE/IFIP Conference on Software Architecture (WICSA'04), 2004, pp. 122–132.

[30] O. Nierstrasz, S. Ducasse, T. Gîrba, The story of Moose: An agile reengineering environment, in: 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13), 2005, pp. 1–10.
[31] H.M. Kienle, Must tool building remain a craft? in: 10th Workshop on Object-Oriented Reengineering (WOOR 2007), 2007. http://scg.unibe.ch/wiki/events/woor2007/.
[32] R. Koschke, Zehn Jahre WSR—Zwölf Jahre Bauhaus, in: 10th Workshop Software Reengineering (WSR 2008), 2008. http://www.informatik.uni-bremen.de/st/papers/bauhaus-wsr08.pdf.
[33] R. Koschke, Atomic Architectural Component Recovery for Program Understanding and Evolution, Ph.D. thesis, University of Stuttgart, Germany, 2000.
[34] R. Ferenc, A. Beszedes, M. Tarkiainen, T. Gyimothy, Columbus—reverse engineering tool an schema for C++, in: 18th IEEE International Conference on Software Maintenance (ICSM'02), 2002, pp. 172–181.
[35] J. Ebert, B. Kullbach, V. Riediger, A. Winter, Gupro—generic understanding of programs: An overview, in: Graph-Based Tools, First International Conference on Graph Transformation, Elsevier, Amsterdam, Vol. 72 of Electronic Notes in Theoretical Computer Science, 2002, pp. 47–56.
[36] P.J. Finnigan, R.C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H.A. Müller, J. Mylopolous, S.G. Perelgut, M. Stanley, K. Wong, The software bookshelf, IBM Syst. J. 36 (4) (1997) 564–593.
[37] M.D. Storey, H.A. Müller, Manipulating and documenting software structures using SHriMP views, in: 11th IEEE International Conference on Software Maintenance (ICSM'95), 1995, pp. 275–284.
[38] H.M. Kienle, Exchange format bibliography, ACM SIGSOFT Softw. Eng. Notes 26 (1) (2001) 56–60.
[39] K. Wong, Rigi User's Manual: Version 5.4.4, Department of Computer Science, University of Victoria, 1998. http://www.rigi.cs.uvic.ca/downloads/rigi/doc/rigi-5.4.4-manual.pdf. (June).
[40] A. Kuhn, T. Verwaest, FAME—a polyglot library for metamodeling at runtime, in: Models@run.time 2008, 2008. http://www.comp.lancs.ac.uk/~bencomo/MRT08/papers/MRT08_manuscript18.pdf.
[41] R.C. Holt, A. Winter, A. Schürr, GXL: Towards a standard exchange format, in: 7th IEEE Working Conference on Reverse Engineering (WCRE'00), 2000, pp. 162–171.
[42] M. Kamp, Managing a Multi-File, Multi-Language Software Repository for Program Comprehension Tools—A Generic Approach, Fachberichte Informatik 1/98, Universität Koblenz-Landau, Institut für Informatik, Koblenz, Germany, 1998.
[43] J. Martin, Leveraging IBM VisualAge for C++ for reverse engineering tasks, in: Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'99), 1999, pp. 83–95.
[44] G.C. Murphy, D. Notkin, Lightweight lexical source model extraction, ACM Trans. Softw. Eng. Methodol. 5 (3) (1996) 262–292.
[45] L. Moonen, Lightweight impact analysis using island grammars, in: 10th IEEE International Workshop on Program Comprehension (IWPC'02), 2002, pp. 219–228.
[46] H. Müller, J. Jahnke, D. Smith, M. Storey, S. Tilley, K. Wong, Reverse engineering: A roadmap, in: Conference on the Future of Software Engineering, 2000, pp. 49–60.
[47] H.A. Müller, H.M. Kienle, Reverse Engineering, in: P. Laplante (Ed.), Encyclopedia of Software Engineering, Auerbach Publications, 2010.
[48] S. Rugaber, Program understanding, in: A. Kent, J.G. Williams (Eds.), Encyclopedia of Computer Science and Technology, Marcel Dekker, New York, 1996, pp. 341–368.
[49] D. Jackson, M. Rinard, Software analysis: A roadmap, in: Conference on the Future of Software Engineering, 2000, pp. 135–145.
[50] H.M. Kienle, H.A. Müller, Leveraging program analysis for web site reverse engineering, in: 3rd IEEE International Workshop on Web Site Evolution (WSE'01), 2001, pp. 117–125.

[51] T. Systä, K. Koskimies, H. Müller, Shimba—an environment for reverse engineering Java software systems, Softw.—Pract. Exp. 31 (4) (2001) 371–394.
[52] M. Hind, Pointer analysis: Haven't we solved this problem yet? in: ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'01), 2001, pp. 54–61.
[53] C. Jones, The Year 2000 Software Problem: Quantifying the Costs and Assessing the Consequences, ACM Press, New York, 1998.
[54] R. Mili, R. Steiner, Software engineering: Introduction, in: S. Diehl (Ed.), Software Visualization, Springer-Verlag, Berlin, Vol. 2269 of Lecture Notes in Computer Science, 2002, pp. 129–137.
[55] G.C. Murphy, D. Notkin, K.J. Sullivan, Software reflection models: Bridging the gap between design and implementation, IEEE Trans. Softw. Eng. 27 (4) (2001) 364–380.
[56] R. Koschke, Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey, J. Softw. Maintenance Evol.: Res. Pract. 15 (2) (2003) 87–109.
[57] M. Lanza, S. Ducasse, Polymetric views—a lightweight visual approach to reverse engineering, IEEE Trans. Softw. Eng. 29 (9) (2003) 782–795.
[58] M.D. Storey, F.D. Fracchia, H.A. Müller, Customizing a fisheye view algorithm to preserve the mental map, J. Vis. Lang. Comput. 10 (3) (1999) 245–267.
[59] J. Michaud, M. Storey, H. Müller, Integrating information sources for visualizing Java programs, in: 17th IEEE International Conference on Software Maintenance (ICSM'01), 2001, pp. 250–258.
[60] S.R. Tilley, M.J. Whitney, H.A. Müller, M.D. Storey, Personalized information structures, in: 11th ACM International Conference on Systems Documentation (SIGDOC'93), 1993, pp. 325–337.
[61] J.S. Jasperson, B.S. Butler, T.A. Carte, H.J.P. Croes, C.S. Saunders, W. Zheng, Power and information technology research: A metatriangulation review, MIS Q. 26 (4) (2002) 397–459.
[62] K. Wong, The reverse engineering notebook, Department of Computer Science, University of Victoria, 1999. http://hdl.handle.net/1828/278. Ph.D. thesis.
[63] L. Bass, M. Klein, F. Bachmann, Quality attribute design primitives and the attribute driven design method, in: F. van der Linden (Ed.), PFE-4 2001, Springer-Verlag, Berlin, Vol. 2290 of Lecture Notes in Computer Science, 2002, pp. 169–186.
[64] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, SEI Series in Software Engineering, Addison-Wesley, Reading, MA, 1998.
[65] D.J. Carney, K.C. Wallnau, A basis for evaluation of commercial software, Inf. Softw. Technol. 40 (14) (1998) 851–860.
[66] S. Tichelaar, Modeling object-oriented software for reverse engineering and refactoring, Ph.D. thesis, Universität Bern, 2001, December.
[67] A. Hamou-Lhadj, T.C. Lethbridge, L. Fu, Challenges and requirements for an effective trace exploration tool, in: 12th IEEE International Workshop on Program Comprehension (IWPC'04), 2004, pp. 70–78.
[68] A. van Deursen, T. Kuipers, Building documentation generators, in: 15th IEEE International Conference on Software Maintenance (ICSM'99), 1999, pp. 40–49.
[69] R. Koschke, J. Girard, M. Würthner, An intermediate representation for integrating reverse engineering analyses, in: 5th IEEE Working Conference on Reverse Engineering (WCRE'98), 1998, pp. 241–250.
[70] S. Ducasse, S. Tichelaar, Dimensions of reengineering environment infrastructures, J. Softw. Maintenance Evol.: Res. Pract. 15 (5) (2003) 345–373.
[71] T. Skramstad, M.K. Khan, Assessment of reverse engineering tools: A MECCA approach, in: 2nd IEEE Symposium on Assessment of Quality Software Development Tools, 1992, pp. 120–126.
[72] B. Bellay, H. Gall, An evaluation of reverse engineering tool capabilities, J. Softw. Maintenance: Res. Pract. 10 (5) (1998) 305–331.

[73] B. Bellay, H. Gall, A comparison of four reverse engineering tools, in: 4th IEEE Working Conference on Reverse Engineering (WCRE'97), 1997, pp. 2–11.
[74] M.N. Armstrong, C. Trudeau, Evaluating architectural extractors, in: 5th IEEE Working Conference on Reverse Engineering (WCRE'98), 1998, pp. 30–39.
[75] M. Sensalire, P. Ogao, A. Telea, Classifying desirable features of software visualization tools for corrective maintenance, in: 4th ACM Symposium on Software Visualization (SOFTVIS'08), 2008, pp. 87–90.
[76] G.C. Murphy, D. Notkin, W.G. Griswold, E.S. Lan, An empirical study of static call graph extractors, ACM Trans. Softw. Eng. Methodol. 7 (2) (1998) 158–191.
[77] S.E. Sim, R.C. Holt, S. Easterbrook, On using a benchmark to evaluate C++ extractors, in: 10th IEEE International Workshop on Program Comprehension (IWPC'02), 2002, pp. 114–123.
[78] Y.-G. Guéhéneuc, K. Mens, R. Wuyts, A comparative framework for design recovery tools, in: 10th IEEE European Conference on Software Maintenance and Reengineering (CSMR'06), 2006, pp. 123–134.
[79] S. Bassil, R.K. Keller, Software visualization tools: Survey and analysis, in: 9th IEEE International Workshop on Program Comprehension (IWPC'01), 2001, pp. 7–17.
[80] H. Padda, A. Seffah, S. Mudur, Investigating the comprehension support for effective visualization tools, in: 2nd International Conference on Advances in Computer-Human Interaction (ACHI'09), 2009, pp. 283–288.
[81] M.J. Pacione, M. Roper, M. Wood, A comparative evaluation of dynamic visualisation tools, in: 10th IEEE Working Conference on Reverse Engineering (WCRE'03), 2003, pp. 80–89.
[82] M.D. Storey, D. Cubranic, D.M. German, On the use of visualization to support awareness of human activities in software development: A survey and a framework, in: ACM Symposium on Software visualization (SoftVis'05), 2005, pp. 193–202.
[83] H.M. Kienle, Building reverse engineering tools with software components, Department of Computer Science, University of Victoria, 2006. https://dspace.library.uvic.ca:8443/dspace/handle/1828/115. Ph.D. thesis (November).
[84] D. West, Looking for love (in all the wrong places), ACM SIGPLAN Notices 39 (12) (2004) 57–63.
[85] G.C. Murphy, D. Notkin, Reengineering with reflexion models: A case study, IEEE Comput. 30 (8) (1997) 29–36.
[86] I.D. Baxter, C. Pidgeon, M. Mehlich, DMS: Program transformations for practical scalable software evolution, in: 26th ACM/IEEE International Conference on Software Engineering (ICSE'04), 2004, pp. 625–634.
[87] J. Favre, J. Estublier, R. Sanlaville, Tool adoption issues in a very large software company, in: 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), 2003, pp. 81–89.
[88] P. Brown, Integrated hypertext and program understanding tools, IBM Syst. J. 30 (3) (1991) 363–392.
[89] S. Ducasse, M. Lanza, S. Tichelaar, Moose: An extensible language-independent environment for reengineering object-oriented systems, in: International Symposium on Constructing Software Engineering Tools (COSET'00), 2000.
[90] T.C. Lethbridge, N. Anquetil, Architecture of a Source Code Exploration Tool: A Software Engineering Case Study, Department of Computer Science, University of Ottawa, Canada, 1997. Tech. Rep. TR-97-07.
[91] S.R. Tilley, Domain-retargetable reverse engineering, Ph.D. thesis, Department of Computer Science, University of Victoria, 1995.

[92] D. Jerding, S. Rugaber, Using visualization for architectural localization and extraction, in: 4th IEEE Working Conference on Reverse Engineering (WCRE'97), 1997, pp. 56–65.
[93] A. Cox, C. Clarke, Representing and accessing extracted information, in: 17th IEEE International Conference on Software Maintenance (ICSM'01), 2001, pp. 12–21.
[94] R. Ferenc, Transformations Among the Three Levels of Schemas for Reverse Engineering, Schloss Dagstuhl, Germany, 2005. http://www.dagstuhl.de/05161/Materials/. Dagstuhl Seminar 05161.
[95] Y. Yu, H. Dayani-Fard, J. Mylopoulos, P. Andritsos, Reducing build time through precompilations for evolving large software, in: 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005, pp. 59–68.
[96] G. St-Denis, R. Schauer, R.K. Keller, Selecting a model interchange format: The SPOOL case study, in: 33rd IEEE Hawaii International Conference on System Sciences (HICSS'00), 2000.
[97] A.S. Yeh, D.R. Harris, M.P. Chase, Manipulating recovered software architecture views, in: 19th ACM/IEEE International Conference on Software Engineering (ICSE'97), 1997, pp. 184–194.
[98] H.M. Sneed, Migrating to web services: A research framework, in: 1st International Workshop on Service-Oriented Architecture Maintenance (SOAM'07), 2007. http://www.cs.vu.nl/csmr2007/workshops/4-%20SneedSOAPaper.pdf.
[99] D.L. Moise, K. Wong, An industrial experience in reverse engineering, in: 10th IEEE Working Conference on Reverse Engineering (WCRE'03), 2003, pp. 275–284.
[100] R. Fiutem, G. Antoniol, P. Tonella, E. Merlo, ART: An architectural reverse engineering environment, J. Softw. Maintenance: Res. Pract. 11 (5) (1999) 339–364.
[101] M. Sensalire, P. Ogao, Visualizing object oriented software: Towards a point of reference for developing tools for industry, in: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'07), 2007, pp. 26–29.
[102] M.D. Storey, K. Wong, P. Fong, D. Hooper, K. Hopkins, H.A. Müller, On designing an experiment to evaluate a reverse engineering tool, in: 3rd IEEE Working Conference on Reverse Engineering (WCRE'96), 1996, pp. 31–40.
[103] J. Czeranski, T. Eisenbarth, H.M. Kienle, R. Koschke, D. Simon, Analyzing xfig using the Bauhaus tool, in: 7th IEEE Working Conference on Reverse Engineering (WCRE'00), 2000, pp. 197–199.
[104] M. Blaha, D. LaPlant, E. Marvak, Requirements for repository software, in: 5th IEEE Working Conference on Reverse Engineering (WCRE'98), 1998, pp. 164–173.
[105] M.W. Godfrey, Practical data exchange for reverse engineering frameworks: Some requirements, some experience, some headaches, ACM SIGSOFT Softw. Eng. Notes 26 (1) (2001) 50–52.
[106] A. Finkelstein, J. Kramer, Software engineering: A roadmap, in: Conference on the Future of Software Engineering, 2000, pp. 5–22.
[107] L.J. Fulop, P. Hegedus, R. Ferenc, T. Gyimóthy, Towards a benchmark for evaluating reverse engineering tools, in: 15th IEEE Working Conference on Reverse Engineering (WCRE'08), 2008, pp. 335–336.
[108] R. Ferenc, S.E. Sim, R.C. Holt, R. Koschke, T. Gyimóthy, Towards a standard schema for C/C++, in: 8th IEEE Working Conference on Reverse Engineering (WCRE'01), 2001, pp. 49–58.
[109] G. Lewis, L. Wrange, Approaches to Constructive Interoperability, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2004. http://www.sei.cmu.edu/pub/documents/04.reports/pdf/04tr020.pdf. Tech. Rep. CMU/SEI-2004- TR-020, December.
[110] M.V. Zelkowitz, Modeling software engineering environment capabilities, J. Syst. Softw. 35 (1) (1996) 3–14.
[111] S. Woods, L. O'Brien, T. Lin, K. Gallagher, A. Quilici, An architecture for interoperable program understanding tools, in: 6th IEEE International Workshop on Program Comprehension (IWPC'98), 1998, pp. 54–63.

[112] R. Kazman, S.G. Woods, S.J. Carrière, Requirements for integrating software architecture and reengineering models: CORUM II, in: 5th IEEE Working Conference on Reverse Engineering (WCRE'98), 1998, pp. 154–163.
[113] M. Zelkowitz, B. Cuthill, Application of an information technology model to software engineering environments, J. Syst. Softw. 37 (1) (1997) 27–40.
[114] R. Kazman, S.J. Carrière, Playing detective: reconstructing software architecture from available evidence, J. Automated Softw. Eng. 6 (2) (1999) 107–138.
[115] J. Froehlich, P. Dourish, Unifying artifacts and activities in a visual tool for distributed software development, in: 26th ACM/IEEE International Conference on Software Engineering (ICSE'04), 2004, pp. 387–396.
[116] T. Panas, J. Lundberg, W. Löwe, Reuse in reverse engineering, in: 12th IEEE International Workshop on Program Comprehension (IWPC'04), 2004, pp. 52–61.
[117] A.W. Brown, Control integration through message-passing in a software development environment, Softw. Eng. J. 8 (3) (1993) 121–131.
[118] D.L. Moise, K. Wong, Issues in integrating schemas for reverse engineering, in: J.-M. Favre, M. Godfrey, A. Winter (Eds.), International Workshop on Meta-Models and Schemas for Reverse Engineering (ateM'03), vol. 94, Elsevier, The Netherlands, 2004, pp. 81–91.
[119] I.T. Bowman, M.W. Godfrey, R.C. Holt, Connecting software architecture recovery frameworks, in: 1st International Symposium on Constructing Software Engineering Tools (CoSET'99), 1999.
[120] T.C. Lethbridge, S. Tichelaar, E. Ploedereder, The dagstuhl middle metamodel: A schema for reverse engineering, in: J.-M. Favre, M. Godfrey, A. Winter (Eds.), International Workshop on Meta-Models and Schemas for Reverse Engineering (ateM'03), vol. 94, Elsevier, The Netherlands, 2004, pp. 7–18.
[121] J. Park, S. Ram, Information systems interoperability: What lies beneath? ACM Trans. Office Inf. Syst. 22 (4) (2004) 595–632.
[122] H.M. Kienle, J. Czeranski, T. Eisenbarth, The API perspective of exchange formats, in: Workshop on Standard Exchange Format (WoSEF), 2000. http://www.ics.uci.edu/~ses/wosef/papers/Kienle-api.ps.
[123] J. Favre, G^{SEE}: A generic software exploration environment, in: 9th IEEE International Workshop on Program Comprehension (IWPC'01), 2001, pp. 233–244.
[124] D. Jin, Ontological Adaptive Integration of Reverse Engineering Tools, Queen's University, Canada, 2004. Ph.D. thesis (August).
[125] S. Meyers, Difficulties in integrating multiple development systems, IEEE Softw. 8 (1) (1991) 49–57.
[126] S.E. Sim, Next generation data interchange: Tool-to-tool application program interfaces, in: 7th IEEE Working Conference on Reverse Engineering (WCRE'00), 2000, pp. 278–280.
[127] G. Ghezzi, H. Gall, Towards software analysis as a service, in: 4th International ERCIM Workshop on Software Evolution and Evolvability (Evol'08), 2008, pp. 1–10.
[128] D.L. Parnas, Designing software for ease of extension and contraction, IEEE Trans. Softw. Eng. SE-5 (2) (1979) 128–137.
[129] J.W. Michaud, A Software Customization Framework, Department of Computer Science, University of Victoria, Victoria, Canada, 2003. Master's thesis.
[130] K.H. Davis, Lessons learned in data reverse engineering, in: 8th IEEE Working Conference on Reverse Engineering (WCRE'01), 2001, pp. 323–327.
[131] C. Best, Designing a Component-Based Framework for a Domain Independent Visualization Tool, Master's thesis, Department of Computer Science, University of Victoria, Victoria, Canada, 2002.

[132] L. Markosian, P. Newcomb, R. Brand, S. Burson, T. Kitzmiller, Using and enabling technology to reengineer legacy systems, Commun. ACM 37 (5) (1994) 58–70.
[133] E. Buss, J. Henshaw, A software reverse engineering experience, in: Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'91), 1991, pp. 55–73.
[134] S.P. Reiss, Constraining software evolution, in: 18th IEEE International Conference on Software Maintenance (ICSM'02), 2002, pp. 162–171.
[135] C. Riva, Reverse architecting: Suggestions for an exchange format, in: Workshop on Standard Exchange Format (WoSEF), 2000.
[136] J. Favre, A new approach to software exploration: Back-packing with G^{SEE}, in: 6th IEEE European Conference on Software Maintenance and Reengineering (CSMR'02), 2002, pp. 251–262.
[137] J.B. Tran, R.C. Holt, Forward and reverse repair of software architecture, in: Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'99), 1999.
[138] P. Newcomb, L. Markosian, Automating the modularization of large cobol programs: Application of an enabling technology for reengineering, in: 1st IEEE Working Conference on Reverse Engineering (WCRE'93), 1993, pp. 222–230.
[139] T.R. Dean, J.R. Cordy, A.J. Malton, K.A. Schneider, Agile parsing in txl, Automated Softw. Eng. 10 (4) (2003) 311–336.
[140] A. Cox, C. Clarke, Syntactic approximation using iterative lexical analysis, in: 11th IEEE International Workshop on Program Comprehension (IWPC'03), 2003, pp. 154–163.
[141] J. Brichau, A. Kellens, S. Castro, T. D'Hondt, Enforcing structural regularities in software using IntensiVE, in: 1st International Workshop on Academic Software Development Tools and Techniques (WASDeTT-1), 2008.
[142] D.C. Atkinson, W.G. Griswold, The design of whole-program analysis tools, in: 18th ACM/IEEE International Conference on Software Engineering (ICSE'96), 1996, pp. 16–27.
[143] M. Eichberg, M. Haupt, M. Mezini, T. Schäfer, Comprehensive software understanding with Sextant, in: 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005, pp. 315–324.
[144] S.P. Reiss, A framework for abstract 3D visualization, in: IEEE Symposium on Visual Languages (VL'93), 1993, pp. 108–115.
[145] S.P. Reiss, An overview of BLOOM, in: ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'01), 2001, pp. 38–45.
[146] Q. Wang, W. Wang, R. Brown, K. Driesen, B. Dufour, EVolve: An open extensible software visualization framework, in: ACM Symposium on Software Visualization (SoftVis'03), 2003, pp. 37–38.
[147] F.J. Newbery, An interface description language for graph editors, in: IEEE Symposium on Visual Languages (VL'88), 1988, pp. 144–149.
[148] E.R. Gansner, S.C. North, An open graph visualization system and its applications to software engineering, Softw.—Pract. Exp. 30 (11) (2000) 1203–1233.
[149] A. Telea, A. Maccari, C. Riva, An open toolkit for prototyping reverse engineering visualizations, in: Symposium on Data Visualization 2002 (VISSYM'02), 2002, pp. 241–249.
[150] M. Meyer, T. Girba, M. Lungu, Mondrian: An agile information visualization framework, in: ACM Symposium on Software visualization (SoftVis'06), 2006, pp. 135–144.
[151] S.E. Sim, M. Storey, A. Winter, A structured demonstration of five program comprehension tools: Lessons learnt, in: 7th IEEE Working Conference on Reverse Engineering (WCRE'00), 2000, pp. 210–212.
[152] A. Marcus, D. Comorski, A. Sergeyev, Supporting the evolution of a software visualization tool through usability studies, in: 13th IEEE International Workshop on Program Comprehension (IWPC'05), 2005, pp. 307–316.

[153] A. Holzinger, Usability engineering methods for software developers, Commun. ACM 48 (1) (2005) 71–74.
[154] E. Folmer, J. Bosch, Architecting for usability: A survey, J. Syst. Softw. 70 (1–2) (2004) 61–78.
[155] H.A. Müller, K. Wong, M. Storey, Reverse engineering research should target cooperative information system requirements, in: 5th IEEE Working Conference on Reverse Engineering (WCRE'98), 1998, p. 255.
[156] A. Walenstein, Cognitive support in software engineering tools: A distributed cognition environment, Ph.D. thesis, Simon Fraser University, Vancouver, 2002, May.
[157] J. Singer, Creating software engineering tools that are useable, useful, and actually used, 2004, talk given at the University of Victoria, December.
[158] A. Maccari, C. Riva, Empirical evaluation of CASE tools usage at Nokia, J. Empirical Softw. Eng. 5 (3) (2000) 287–299.
[159] N. Bevan, Quality in use: Meeting user needs for quality, J. Syst. Softw. 49 (1) (1999) 89–96.
[160] M.A. Toleman, J. Welsh, Systematic evaluation of design choices for software development tools, Softw.—Concepts & Tools 19 (3) (1998) 109–121.
[161] S.L. Smith, J.N. Mosier, Guidelines for Designing User Interface Software, The MITRE Corporation, Bedford, MA, 1986. http://www.dfki.de/~jameson/hcida/papers/smith-mosier.pdf. Tech. Rep. ESD-TR-86-278, August.
[162] S.P. Reiss, Incremental maintenance of software artifacts, in: 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005, pp. 113–122.
[163] M.-A.D. Storey, A Cognitive Framework for Describing and Evaluating Software Exploration Tools, Ph.D. thesis, Simon Fraser University, Vancouver, 1998, December.
[164] M.D. Storey, F.D. Fracchia, H.A. Müller, Cognitive design elements to support the construction of a mental model during software exploration, J. Syst. Softw. 44 (3) (1999) 171–185.
[165] L. Bass, B.E. John, J. Kates, Achieving Usability Through Software Architecture, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2001. http://www.sei.cmu.edu/pub/documents/01.reports/pdf/01tr005.pdf. Tech. Rep. CMU/SEI-2001-TR-005, March.
[166] S.P. Reiss, The paradox of software visualization, in: 3rd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'05), 2005, pp. 59–63.
[167] R.I. Bull, A. Trevors, A.J. Malton, M.W. Godfrey, Semantic grep: Regular expressions + relational abstraction, in: 9th IEEE Working Conference on Reverse Engineering (WCRE'02), 2002, pp. 267–276.
[168] S. Paul, A. Prakash, A query algebra for program databases, IEEE Trans. Softw. Eng. 22 (3) (1996) 202–217.
[169] M. D'Ambros, M. Lanza, M. Lungu, R. Robbes, Promises and perils of porting software visualization tool to the web, in: 11th IEEE International Symposium on Web Systems Evolution (WSE'09), 2009.
[170] S. Manchoridis, T.S. Sounder, Y. Chen, D.R. Gansner, J.L. Korn, REportal: A web-based portal site for reverse engineering, in: 8th IEEE Working Conference on Reverse Engineering (WCRE'01), 2001, pp. 221–230.
[171] T.C. Lethbridge, J. Singer, Strategies for studying maintenance, in: 2nd Workshop on Empirical Studies of Software Maintenance (WESS'96), 1996, pp. 79–83.
[172] J.R. Cordy, Comprehending reality—Practical barriers to industrial adoption of software maintenance automation, in: 11th IEEE International Workshop on Program Comprehension (IWPC'03), 2003, pp. 196–206.
[173] M.D. Storey, K. Wong, H.A. Müller, How do program understanding tools affect how programmers understand programs, in: 4th IEEE Working Conference on Reverse Engineering (WCRE'97), 1997, pp. 12–21.

[174] J. Singer, T. Lethbridge, Studying work practices to assist tool design in software engineering, in: 6th IEEE International Workshop on Program Comprehension (IWPC'98), 1998, pp. 173–179.
[175] T.C. Lethbridge, F. Herrera, Assessing the usefulness of the TKSee software exploration tool, in: H. Erdogmus, O. Tanir (Eds.), Advances in Software Engineering: Topics in Comprehension, Evolution, and Evaluation, Springer-Verlag, Berlin, 2001, pp. 73–93, Ch. 11.
[176] J. Grudin, Interactive systems: Bridging the gaps between developers and users, IEEE Comput. 24 (4) (1991) 59–69.
[177] T. Schäfer, M. Eichberg, M. Haupt, M. Mezini, The SEXTANT software exploration tool, IEEE Trans. Softw. Eng. 32 (9) (2006) 753–768.
[178] T.R.G. Green, M. Petre, Usability analysis of visual programming environments: A 'cognitive dimensions' framework, J. Vis. Lang. Comput. 7 (2) (1996) 131–174.
[179] L. Heinz, TransPlant: Helping organizations to make the transition, news@sei 4 (4) 2002. http://sei.cmu.edu/news-at-sei/features/2001/4q01/feature-4-4q01.html.
[180] E.M. Rogers, Diffusion of Innovations, fourth Ed., The Free Press, New York, 1995.
[181] J. Iivari, Why are CASE tools not used? Commun. ACM 39 (10) (1996) 94–103.
[182] R. Holt, Wosef introduction, in: Workshop on Standard Exchange Format (WoSEF), 2000.
[183] E. Buss, J. Henshaw, Experiences in program understanding, in: Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'92), 1992, pp. 55–73.
[184] R. Kollmann, Experiences with an industrial long-term reengineering project, in: 11th IEEE Working Conference on Reverse Engineering (WCRE'04), 2004, pp. 8–16.
[185] A. von Mayrhauser, A.M. Vans, From program comprehension to tool requirements for an industrial environment, in: 2nd IEEE Workshop on Program Comprehension (WPC'93), 1993, pp. 78–86.
[186] S. Rugaber, L.M. Wills, Creating a research infrastructure for reengineering, in: 3rd IEEE Working Conference on Reverse Engineering (WCRE'96), 1996, pp. 98–102.
[187] B. Balzer, M. Litoiu, H. Müller, D. Smith, M. Storey, S. Tilley, K. Wong, 4th international workshop on adoption-centric software engineering, in: 4th International Workshop on Adoption-Centric Software Engineering (ACSE'04), 2004, pp. 1–2.
[188] T.C. Lethbridge, Value assessment by potential tool adopters: Towards a model that considers costs, benefits and risks of adoption, in: 4th International Workshop on Adoption-Centric Software Engineering (ACSE'04), 2004, pp. 46–50.
[189] S. Tilley, S. Huang, T. Payne, On the challenges of adopting ROTS software, in: 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), 2003, pp. 3–6.
[190] S. Huang, S. Tilley, Z. Zhiying, On the yin and yang of academic research and industrial practice, in: 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), 2003, pp. 19–22.
[191] S. Tilley, S. Huang, On selecting software visualization tools for program understanding in an industrial context, in: 10th IEEE International Workshop on Program Comprehension (IWPC'02), 2002, pp. 285–288.
[192] K. Wong, On inserting program understanding technology into the software change process, in: 4th IEEE Workshop on Program Comprehension (WPC'96), 1996, pp. 90–99.
[193] P.T. Devanbu, Re-targetability in software tools, ACM SIGAPP Appl. Comput. Rev. 7 (3) (1999) 19–26.
[194] J. Martin, Tool adoption: A software developer's perspective, in: 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), 2003, pp. 7–9.
[195] S.R. Tilley, Coming attractions in program understanding ii: Highlights of 1997 and opportunities in 1998, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1998. http://

www.sei.cmu.edu/pub/documents/98.reports/pdf/98tr001.pdf. Technical Report CMU/SEI-98-TR-001.
[196] M. Jazayeri, The education of a software engineer, in: 19th IEEE Conference on Automated Software Engineering (ASE'04), 2004, pp. xviii–xxvii.
[197] T.C. Lethbridge, J. Singer, Understanding software maintenance tools: Some empirical research, in: 3rd Workshop on Empirical Studies of Software Maintenance (WESS'97), 1997, pp. 157–162.
[198] R.L. Glass, Facts and Fallacies of Software Engineering, Addison-Wesley, Boston, MA, 2002.
[199] D.A. Patterson, 20th century vs. 21st century c&c: The SPUR manifesto, Commun. ACM 48 (3) (2005) 15–16.
[200] S. Tilley, D. Distante, On the adoption of an approach to reengineering web application transactions, in: 4th International Workshop on Adoption-Centric Software Engineering (ACSE'04), 2004, pp. 13–20.
[201] S. Rifkin, Two good reasons why new software processes are not adopted, in: 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), 2003, pp. 23–29.
[202] S. Huang, S. Tilley, On the challenges in fostering adoption via empirical studies, in: 4th International Workshop on Adoption-Centric Software Engineering (ACSE'04), 2004, pp. 43–45.
[203] M. Storey, Theories, methods and tools in program comprehension: Past, present and future, in: 13th IEEE International Workshop on Program Comprehension (IWPC'05), 2005, pp. 181–191.
[204] A. Walenstein, Observing and measuring cognitive support: Steps toward systematic tool evaluation and engineering, in: 11th IEEE International Workshop on Program Comprehension (IWPC'03), 2003, pp. 185–194.
[205] D. Sun, K. Wong, On understanding software tool adoption using perceptual theories, in: 4th International Workshop on Adoption-Centric Software Engineering (ACSE'04), 2004, pp. 51–55.
[206] C. Krueger, Eliminating the adoption barrier, IEEE Softw. 19 (4) (2002) 29–31.
[207] H. Wang, C. Wang, Open source software adoption: A status report, IEEE Softw. 18 (2) (2001) 90–95.
[208] J. Grudin, Groupware and social dynamics: Eight challenges for developers, Commun. ACM 37 (1) (1994) 92–105.
[209] A. Moran, Report on the first commercial users of functional programming workshop, ACM SIGPLAN Notices 39 (12) (2004) 17–20.
[210] W. Harrison, R. Wieringa, Introduction to the workshop on technology transfer in software engineering, in: 2006 International Workshop on Software Technology Transfer in Software Engineering (TT'06), 2006, pp. 1–2.
[211] J. Segal, C. Morris, Developing scientific software, part 2, IEEE Softw. 26 (1) (2009) 79.
[212] H.M. Kienle, H.A. Müller, Requirements of software visualization tools: A literature survey, in: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007), 2007, pp. 2–9.
[213] S.E. Sim, R. Koschke, Wosef: Workshop on standard exchange format, ACM SIGSOFT Softw. Eng. Notes 26 (1) (2001) 44–49.
[214] S. Tichelaar, S. Ducasse, S. Demeyer, FAMIX: Exchange experiences with CDIF and XMI, in: Workshop on Standard Exchange Format (WoSEF), 2000. http://www.ics.uci.edu/~ses/wosef/papers/Tichelaar.ps.
[215] R.C. Holt, A. Winter, J. Wu, Towards a Common Query Language for Reverse Engineering, Institut für Informatik, Universität Koblenz-Landau, Koblenz, Germany, 2002. Tech. Rep. 8/2002, August.
[216] A. Bosworth, Learning from the web, ACM Queue 3 (8) (2005) 26–32.

[217] J. Mendling, A survey on design criteria for interchange formats, Department of Information Systems, New Media Lab, Vienna University of Economics and Business Administration, Vienna, Austria, 2004. Technical Report.
[218] U. Brandes, M. Eiglsperger, I. Merman, M. Himsolt, M.S. Marshall, GraphML progress report: Structural layer proposal, in: 9th International Symposium on Graph Drawing (GD 2001), 2001, pp. 109–112.
[219] S.R. Tilley, D.B. Smith, On using the web as infrastructure for reengineering, in: 5th IEEE International Workshop on Program Comprehension (IWPC'97), 1997, pp. 170–173.
[220] J. Buckley, Requirements-based visualization tools for software maintenance and evolution, IEEE Comput. 42 (4) (2009) 106–108.
[221] D.B. Smith, P.W. Oman, Software tools in context, IEEE Softw. 7 (3) (1990) 15–19.
[222] R. Koschke, Software visualization for reverse engineering, in: S. Diehl (Ed.), Software Visualization, Springer-Verlag, Berlin, Vol. 2269 of Lecture Notes in Computer Science, 2002, pp. 138–150.
[223] S. Hupfer, L.-T. Cheng, S. Ross, J. Patterson, Introducing collaboration into an application development environment, in: ACM Conference on Computer Supported Cooperative Work (CSCW'04), 2004, pp. 21–24.
[224] M. D'Ambros, M. Lanza, A flexible framework to support collaborative software evolution analysis, in: 12th IEEE European Conference on Software Maintenance and Reengineering (CSMR'08), 2008, pp. 3–12.
[225] M. Shaw, Architectural issues in software reuse: It's not just the functionality, it's the packaging, in: Symposium on Software Reusability (SSR'95), 1995, pp. 3–6.
[226] C.W. Krueger, Software reuse, ACM Comput. Surv. 24 (2) (1992) 131–183.
[227] M. Shaw, D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, New Jersey, 1996.
[228] A.I. Wasserman, Tool integration in software engineering environments, in: F. Long (Ed.), Software Engineering Environments: International Workshop on Environments, Springer-Verlag, Berlin, Vol. 467 of Lecture Notes in Computer Science, 1990, pp. 137–149.
[229] P.H. Salus, A Quarter Century of UNIX, UNIX and Open Systems Series, Addison-Wesley, Boston, MA, 1994.
[230] B. Krishnamurthy, Practical Reusable UNIX Software, Wiley, New York, NY, 1995.
[231] C. Best, M. Storey, J. Michaud, Designing a component-based framework for visualization in software engineering and knowledge engineering, in: 14th ACM/IEEE International Conference on Software Engineering and Knowledge Engineering (SEKE'02), 2002, pp. 323–326.
[232] A. Telea, L. Voinea, An interactive reverse engineering environment for large-scale C++ code, in: 4th ACM Symposium on Software Visualization (SOFTVIS'08), 2008, pp. 67–76.
[233] H. Reubenstein, R. Piazza, S. Roberts, Separating parsing and analysis in reverse engineering, in: 1st IEEE Working Conference on Reverse Engineering (WCRE'93), 1993, pp. 117–125.
[234] R. Biddle, A. Martin, J. Noble, No name: Just notes on software reuse, ACM SIGPLAN Notices 38 (2) (2004) 76–96.
[235] H.M. Kienle, Component-based tool development, in: Frontiers of Software Maintenance (FoSM 2008) at ICSM 2008, 2008, pp. 87–98.
[236] S.P. Reiss, The Desert environment, ACM Trans. Softw. Eng. Methodol. 8 (4) (1999) 297–342.
[237] A.L. Powell, J.C. French, J.C. Knight, A systematic approach to creating and maintaining software documentation, in: 11th ACM Symposium on Applied Computing (SAC'96), 1996, pp. 201–208.
[238] K. Czarnecki, U.W. Eisenecker, Generative Programming: Methods, Tools, and Applications, Addison-Wesley, Boston, MA, 2000.

[239] Q. Zhu, Y. Chen, P. Kaminski, A. Weber, H. Kienle, H.A. Müller, Leveraging Visio for adoption-centric reverse engineering tools, in: 10th IEEE Working Conference on Reverse Engineering (WCRE'03), 2003, pp. 270–274.
[240] J. Hartmann, S. Huang, S. Tilley, Documenting software systems with views II: An integrated approach based on XML, in: 19th ACM International Conference on Computer Documentation (SIGDOC'01), 2001, pp. 237–246.
[241] C. Riva, Y. Yang, Generation of architectural documentation using XML, in: 9th IEEE Working Conference on Reverse Engineering (WCRE'02), 2002, pp. 161–169.
[242] N.M. Goldman, R.M. Balzer, The ISI visual design editor generator, in: IEEE Symposium on Visual Languages (VL'99), 1999, pp. 20–27. http://mr.teknowledge.com/daml/briefing_associate_publications.htm.
[243] M.A. Copenhafer, K.J. Sullivan, Exploration harnesses: Tool-supported interactive discovery of commercial component properties, in: 14th IEEE Conference on Automated Software Engineering (ASE'99), 1999, pp. 7–14.
[244] D. Coppit, K.J. Sullivan, Galileo: A tool built from mass-market applications, in: 22nd ACM/IEEE International Conference on Software Engineering (ICSE'00), 2000, pp. 750–753.
[245] D. Coppit, K.J. Sullivan, Sound methods and effective tools for engineering modeling and analysis, in: 25th ACM/IEEE International Conference on Software Engineering (ICSE'03), 2003, pp. 198–207.
[246] S.P. Reiss, Program editing in a software development environment (draft). http://www.cs.brown.edu/~spr/research/desert/fredpaper.pdf, 1995.
[247] X. Wu, A. Murray, M. Storey, R. Lintern, A reverse engineering approach to support software maintenance: Version control knowledge extraction, in: 11th IEEE Working Conference on Reverse Engineering (WCRE'04), 2004, pp. 90–99.
[248] F. Arcelli, C. Tosi, M. Zanoni, S. Maggioni, The MARPLE project—A tool for design pattern detection and software architecture reconstruction, in: 1st International Workshop on Advanced Software Development Tools and Techniques (WASDeTT-1), 2008. http://scg.unibe.ch/download/wasdett/wasdett2008-paper02.pdf.
[249] A. Egyed, P.B. Kruchten, Rose/Architect: A tool to visualize architecture, in: 32rd IEEE Hawaii International Conference on System Sciences (HICSS'99), 1999.
[250] A.F. Egyed, Heterogeneous View Integration and its Automation, University of Southern California, Los Angeles, CA, 2000. Ph.D. thesis, August.
[251] B. Berenbach, Towards a unified model for requirements engineering, in: 4th International Workshop on Adoption-Centric Software Engineering (ACSE'04), 2004, pp. 26–29.
[252] K. Mehner, Javis: A UML-based visualization and debugging environment for concurrent Java programs, in: S. Diehl (Ed.), Software Visualization, Springer-Verlag, Berlin, Vol. 2269 of Lecture Notes in Computer Science, 2002, pp. 163–175.
[253] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, P. Mutzel, A new approach for visualizing UML class diagrams, in: ACM Symposium on Software Visualization (SoftVis'03), 2003, pp. 179–188.
[254] F. Ricca, P. Tonella, Understanding and restructuring Web sites with ReWeb, IEEE MultiMed. 8 (2) (2001) 40–51.
[255] G. Antoniol, R. Fiutem, G. Lutteri, P. Tonella, S. Zanfei, E. Merlo, Program understanding and maintenance with the CANTO environment, in: 13th IEEE International Conference on Software Maintenance (ICSM'97), 1997, pp. 72–81.
[256] T.Y. Lin, F. Zou, H.M. Kienle, H.A. Müller, A customizable SVG graph visualization engine, in: SVG Open 2007, 2007. http://www.svgopen.org/2007/papers/CustomizableSVGGraphVisualizationEngine/.

[257] M. Lungu, M. Lanza, The small project observatory, in: 1st International Workshop on Advanced Software Development Tools and Techniques (WASDeTT-1), 2008. http://scg.unibe.ch/download/wasdett/wasdett2008-paper08.pdf.
[258] D. Holten, Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data, IEEE Trans. Vis. Comput. Graph. 12 (5) (2006) 741–748.
[259] R. Wettel, M. Lanza, Visualizing software systems as cities, in: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'07), 2007, pp. 92–99.
[260] D. Garlan, R. Allen, J. Ockerbloom, Architectural mismatch or why it's hard to build systems out of existing parts, in: 17th ACM/IEEE International Conference on Software Engineering (ICSE'95), 1995, pp. 179–185.
[261] D. Rayside, M. Litoiu, M. Storey, C. Best, R. Lintern, Visualizing flow diagrams in WebSphere Studio using SHriMP views, Inf. Syst. Front. 5 (2) (2003) 161–174.
[262] A. Egyed, R. Balzer, Unfriendly COTS integration—Instrumentation and interfaces for improved plugability, in: 16th IEEE Conference of Automated Software Engineering (ASE'01), 2001, pp. 223–231.
[263] S. Kelly, J.-P. Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation, Wiley, New Jersey, 2008.
[264] T.J. Biggerstaff, A perspective of generative reuse, Ann. Softw. Eng. 5 (1998) 169–226.
[265] S. Rugaber, K. Stirewalt, Model-driven reverse engineering, IEEE Softw. 21 (4) (2004) 45–53.
[266] J. Favre, Cacophony: Metamodel-driven software architecture reconstruction, in: 11th IEEE Working Conference on Reverse Engineering (WCRE'04), 2004, pp. 204–213.
[267] S. Ducasse, T. Gîrba, A. Kuhn, L. Renggli, Meta-environment and executable meta-language using smalltalk: An experience report, J. Softw. Syst. Model. 8 (1) (2009) 5–19.
[268] S. Ducasse, T. Gîrba, Using smalltalk as a reflective executable meta-language, in: 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2006), 2006, pp. 604–618.
[269] K. Cremer, A. Marburger, B. Westfechtel, Graph-based tools for re-engineering, J. Softw. Maintenance and Evol.: Res. Pract. 14 (4) (2002) 257–292.
[270] R.I. Bull, M.-A. Storey, J.-M. Favre, M. Litoiu, An architecture to support model driven software visualization, in: 14th IEEE International Conference on Program Comprehension (ICPC'06), 2006, pp. 100–106.
[271] R.I. Bull, Model Driven Visualization: Towards a Model Driven Engineering Approach for Information Visualization, Department of Computer Science, University of Victoria, Melbourne, Australia, 2008. http://hdl.handle.net/1828/1048. Ph.D. thesis.
[272] D. Coppit, K.J. Sullivan, Multiple mass-market applications as components, in: 22nd ACM/IEEE International Conference on Software Engineering (ICSE'00), 2000, pp. 273–282.
[273] M. Lanza, Codecrawler—Lessons learned in building a software visualization tool, in: 7th IEEE European Conference on Software Maintenance and Reengineering (CSMR'03), 2003, pp. 1–10.
[274] Y.-G. Guéhéneuc, Ptidej: Promoting patterns with patterns, in: 1st ECOOP workshop on Building a System using Patterns, 2005. http://www.iro.umontreal.ca/~ptidej/Publications/Documents/ECOOP05BSUP.doc.pdf.
[275] R. Balzer, A. Egyed, N. Goldman, T. Hollebeek, M. Tallis, D. Wile, Adapting COTS applications: An experience report, in: 2nd IEEE International Workshop on Incorporating COTS-Software into Software Systems: Tools and Techniques (IWICSS'07), 2007.
[276] H.M. Kienle, Building reverse engineering tools with software components: Ten lessons learned, in: 14th IEEE Working Conference on Reverse Engineering (WCRE 2007), 2007, pp. 289–292.

[277] R. Wuyts, S. Ducasse, Unanticipated integration of development tools using the classification model, Computer Languages, Syst. Struct. 30 (1–2) (2004) 63–77.
[278] P. Klint, R. Lämmel, C. Verhoef, Toward an engineering discipline for grammarware, ACM Trans. Softw. Eng. Methodol. 14 (3) (2005) 331–380.
[279] H.M. Kienle, H.A. Müller, Towards a process for developing maintenance tools in academia, in: 15th IEEE Working Conference on Reverse Engineering (WCRE'08), 2008, pp. 237–246.
[280] R. Wuyts, H.M. Kienle, K. Mens, M. van den Brand, A. Kuhn, Academic software development tools and techniques: Report on the 1st workshop WASDeTT at ECOOP 2008, in: P. Eugster (Ed.), Object-Oriented Technology, Springer Verlag, Berlin, ECOOP 2008 Workshop Reader, Vol. 5475 of Lecture Notes in Computer Science, 2009, pp. 87–103. .
[281] H.M. Kienle, L. Moonen, M.W. Godfrey, H.A. Müller, 2nd international workshop on advanced software development tools and techniques (WASDeTT): Tools for software maintenance, visualization, and reverse engineering, in: 24th IEEE International Conference on Software Maintenance (ICSM'08), 2008, pp. 408–409.
[282] M. van den Brand, Guest editor's introduction: Experimental software and toolkits (EST), Sci. Comput. Program. 69 (1–3) (2007) 1–2.
[283] D.I.K. Sjoberg, T. Dyba, M. Jorgensen, Collaboration in software engineering: A roadmap, in: Future of Software Engineering (FOSE'07), 2007, pp. 214–225.
[284] S.E. Sim, A Theory of Benchmarking with Applications to Software Reverse Engineering, University of Toronto, Ontario, Canada, 2003. Ph.D. thesis, October.
[285] B. Kitchenham, DESMET: A Method for Evaluating Software Engineering Methods and Tools, Department of Computer Science, University of Keele, Staffordshire, UK, 1996. Tech. Rep. TR96-09.
[286] T. Panas, M. Staron, Evaluation of a framework for reverse engineering tool construction, in: 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005, pp. 145–154.
[287] S.E. Sim, S. Easterbrook, R.C. Holt, Using benchmarking to advance research: A challenge to software engineering, in: 25th ACM/IEEE International Conference on Software Engineering (ICSE'03), 2003, pp. 74–83.
[288] S. Demeyer, T. Mens, M. Wermelinger, Towards a software evolution benchmark, in: 4th ACM International Workshop on Principles of Software Evolution (IWPSE'04), 2001, pp. 174–177.
[289] A. Lakhotia, J. Li, A. Walenstein, Y. Yang, Towards a clone detection benchmark suite and results archive, in: 11th IEEE International Workshop on Program Comprehension (IWPC'03), 2003, pp. 285–286.
[290] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, Comparison and evaluation of clone detection tools, IEEE Trans. Softw. Eng. 33 (9) (2007) 577–591.
[291] L.J. Fulop, R. Ferenc, T. Gyimothy, Towards a benchmark for evaluating design pattern miner tools, in: 12th IEEE European Conference on Software Maintenance and Reengineering (CSMR'08), 2008, pp. 143–152.
[292] R. Robbes, Of Change and Software, Ph.D. thesis, University of Lugano, Lugano, Switzerland, 2008, December.
[293] S.E. Sim, M. Storey, A structured demonstration of five program comprehension tools, in: 7th IEEE Working Conference on Reverse Engineering (WCRE'00), 2000, pp. 184–193.
[294] J. Charles, Snapshots from industry: Reverse engineering project seeking participants, IEEE Softw. 14 (3) (1997) 118.
[295] M.D. Storey, S.E. Sim, K. Wong, A collaborative demonstration of reverse engineering tools, ACM SIGAPP Appl. Comput. Rev. 10 (1) (2002) 18–25.

[296] M. Sensalire, P. Ogao, A. Telea, Evaluation of software visualization tools: Lessons learned, in: 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'09), 2009.
[297] G. Confora, M. Di Penta, New frontiers of reverse engineering, in: Future of Software Engineering (FOSE'07), 2007, pp. 326–341.
[298] G.A.D. Lucca, M.D. Penta, Experimental settings in program comprehension: Challenges and open issues, in: 14th IEEE International Conference on Program Comprehension (ICPC'06), 2006.
[299] D.I. Sjoberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.-K. Liborg, A.C. Rekdal, A survey of controlled experiments in software engineering, IEEE Trans. Softw. Eng. 31 (9) (2005) 733–753.
[300] M.P. O'Brien, J. Buckley, C. Exton, Empirically studying software practitioners—Bridging the gap between theory and practice, in: 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005, pp. 433–442.
[301] E. Arisholm, H. Gallis, T. Dyba, D.I. Sjoberg, Evaluating pair programming with respect to system complexity and programmer expertise, IEEE Trans. Softw. Eng. 33 (2) (2007) 65–86.
[302] J.E. Sieber, Protecting research subjects, employees and researchers: Implications for software engineering, J. Empirical Softw. Eng. 6 (4) (2001) 329–341.
[303] D.M. Berry, W.F. Tichy, Comments on "formal methods application: An empirical tale of software development" IEEE Trans. Softw. Eng. 29 (6) (2003) 567–571.
[304] L. Hattori, M. Lanza, An environment for synchronous software development, in: ICSE Companion, 31st International Conference on Software Engineering (ICSE 2009), 2009, pp. 223–226.
[305] I. Au, R. Boardman, R. Jeffries, P. Larvie, A. Pavese, J. Riegelsberger, K. Rodden, M. Stevens, User experience at Google: Focus on the user and all else will follow, in: Conference on Human Factors in Computing Systems (CHI'08), 2008, pp. 3681–3686.
[306] D.E. Knuth, Theory and practice, Theor. Comput. Sci. 90 (1) (1991) 1–15.
[307] C. Exton, Constructivism and program comprehension strategies, in: 10th IEEE International Workshop on Program Comprehension (IWPC'02), 2002, pp. 281–284.
[308] E. Soloway, K. Ehrlich, Empirical studies of programming knowledge, IEEE Trans. Softw. Eng. SE-10 (5) (1984) 595–609.
[309] T.C. Lethbridge, S.E. Sim, J. Singer, Studying software engineers: Data collection techniques for software field studies, J. Empirical Softw. Eng. 10 (3) (2005) 311–341.
[310] A. von Mayrhauser, A.M. Vans, From code understanding needs to reverse engineering tool capabilities, in: 5th IEEE International Workshop on Computer-Aided Software Engineering (CASE'93), 1993, pp. 230–239.
[311] A.J. Ko, H.H. Aung, B.A. Myers, Eliciting design requirements for maintenance-oriented ides: A detailed study of corrective and perfective maintenance tasks, in: 27th ACM/IEEE International Conference on Software Engineering (ICSE'05), 2005, pp. 126–135.
[312] A. Walenstein, HASTI: A lightweight framework for cognitive reengineering analysis, in: 14th Psychology of Programmers Workshop (PPIG'02), 2002. http://www.ppig.org/papers/14th-walenstein.pdf.
[313] A.R. Murray, Discourse structure of software explanation: Snapshot theory, cognitive patterns and grounded theory methods, University of Ottawa, Ontario, Canada, 2006. Ph.D. thesis.
[314] A. Murray, T.C. Lethbridge, Presenting micro-theories of program comprehension in pattern form, in: 13th IEEE International Workshop on Program Comprehension (IWPC'05), 2005, pp. 45–54.
[315] H. Farah, Applying Cognitive Patterns to Support Software Tool Development, Ph.D. thesis, University of Ottawa, Ontario, Canada, 2006.

[316] A. Murray, T.C. Lethbridge, On generating cognitive patterns of sofware comprehension, in: Conference of The Centre for Advanced Studies On Collaborative Research (CASCON'05), 2005, pp. 200–211.
[317] S. Adolph, W. Hall, P. Kruchten, A methodological leg to stand on: Lessons learned using grounded theory to study software development, in: Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'08), 2008, pp. 166–178.
[318] C. Hoare, Retrospective: An axiomatic basis for computer programming, Commun. ACM 52 (10) (2009) 30–32.
[319] A. Sillitti, Designing empirical studies: Assessing the effectiveness of agile methods, ACM SIGSOFT Softw. Eng. Notes 34 (5) (2009) 35–37.
[320] D.L. Parnas, The limits of empirical studies of software engineering, in: IEEE International Symposium on Empirical Software Engineering (ISESE'03), 2003, pp. 2–5.
[321] L.C. Briand, The experimental paradigm in reverse engineering: Role, challenges, and limitations, in: 13th IEEE Working Conference on Reverse Engineering (WCRE'06), 2006, pp. 3–8.
[322] M. Di Penta, R. Stirewalt, E. Kraemer, Designing your next empirical study on program comprehension, in: 15th IEEE International Conference on Program Comprehension (ICPC'07), 2007, pp. 281–285.
[323] S. Lang, A. von Mayrhauser, Building a research infrastructure for program comprehension observations, in: 5th IEEE International Workshop on Program Comprehension (IWPC'97), 1997, pp. 165–169.
[324] D. Budgen, G. Hoffnagle, M. Muller, F. Robert, A. Sellami, S. Tilley, Empirical software engineering: A roadmap, in: 10th International Workshop on Software Technology and Engineering Practice (STEP'02), 2002.
[325] M.V. Zelkowitz, Data sharing enabling technologies: Working group results, in: V.R. Basili et al., (Ed.), Empirical Software Engineering Issues, Springer-Verlag, Berlin, Vol. 4336 of Lecture Notes in Computer Science, 2007, pp. 108–110.
[326] M. Oivo, New opportunities for empirical research, in: V.R. Basili et al. (Ed.), Empirical Software Engineering Issues, Springer-Verlag, Berlin, 2007, p. 22. Vol. 4336 of Lecture Notes in Computer Science.
[327] O. Nierstrasz, M. Denker, T. Gîrba, A. Kuhn, A. Lienhard, D. Röthlisberger, Self-Aware, Evolving Eternal Systems, Universtiy of Bern, Switzerland, 2008. http://www.iam.unibe.ch/publikationen/techreports/2008/iam-08-001. Technischer Bericht IAM-08-001.
[328] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, M. Jazayeri, Challenges in software evolution, in: 8th IEEE International Workshop on Principles of Software Evolution (IWPSE'05), 2005, pp. 13–22.
[329] H.A. Müller, H.M. Kienle, U. Stege, Autonomic computing: Now you see it, now you don't, in: A.D. Lucia, F. Ferrucci (Eds.), Software Engineering: International Summer Schools, Springer Verlag, Berlin, ISSSE 2006–2008, Salerno, Italy, Revised Tutorial Lectures, Vol. 5413 of Lecture Notes in Computer Science, 2009, pp. 32–54.
[330] H. Müller, Bits of history, challenges for the future and autonomic computing technology, in: 13th IEEE Working Conference on Reverse Engineering (WCRE'06), 2006, pp. 9–18.
[331] D. Smith, E. Morris, D. Carney, Adoption centric problems in the context of systems of systems interoperability, in: 4th International Workshop on Adoption-Centric Software Engineering (ACSE'04), 2004, pp. 63–68.
[332] W. Pollak (Ed.), Ultra-Large-Scale Systems: The Software Challenge of the Future, SEI, Pittsburgh, PA, 2006. http://www.sei.cmu.edu/uls/.

Author Index

A

Abdullah, K., 42
Ackerman, T. P., 50
Adams, K., 177
Adkins, J. N., 30, 35–36
Adl-Tabatabai, A., 82
Admon, A., 32
Adolph, S., 268, 270
Adve, S. V., 104
Aebersold, R., 28, 36
Agarwal, A., 113
Ahamad, M., 42
Aho, A. V., 192, 244
AI-Mohanna, M., 36
Aingaran, K., 10
Ajwani, D., 13
Akera, A., 73
Akhter, S., 87, 104
Ala'illma, T. F., 73
Alaiya, A., 36
Allen, R., 98–99, 251
Almquist, J., 17
Almquist, J. P., 18, 64
Aloise, F., 178
Altintas, I., 17, 58
Altschul, S. F., 21
Andersen, R. A., 177
Anderson, G. A., 6, 36
Anderson, I., 23
Anderson, K. S., 36
Anderson, L., 36
Anderson, S. L., 36
Anderson, W., 10
Andrews-Pfannkoch, C., 5, 29
Andritsos, P., 203
Ang, L. P., 73
Anquetil, N., 193, 202, 206–207, 214, 236, 241
Anthes, G., 5
Antoniol, G., 204, 250, 260
Antonopoulous, C. D., 9
Antony, R., 137
Arbaugh, W., 121, 134–135, 143
Arcelli, F., 250
Ardekani, A. M., 36
Arisholm, E., 264
Armstrong, M. N., 201, 204–206, 213, 241
Arnaldi, B., 174
Arnold, R. S., 191
Asano, S., 9
Asanovic, K., 78, 80
Aspray, W., 73
Atkinson, D. C., 216
Au, I., 264
Auberry, K. J., 36
Aung, H. H., 267, 273

B

Babiloni, F., 178
Bafna, V., 29, 36
Bahli, B., 121, 127
Bahrmann, C., 56
Bakay, R. A. E., 177
Baker, B., 149
Balzer, B., 230–231, 244
Balzer, R. M., 250–252, 256

Barabasi, A.-L., 6
Barnea, E., 32
Barret, R., 80
Barry, R. C., 29, 36
Bashashati, A., 176
Bass, L., 200, 223, 243–244, 265
Bassil, S., 201, 205, 216, 222–223, 235
Batmaz, I., 136
Baxter, D. J., 36
Baxter, I. D., 201, 204
Bayliss, J. D., 172
Beavis, R. C., 31–32, 36
Beck, P. D., 177
Beer, I., 32
Behrens, B. C., 190
Behrens, F., 36
Bell, C. J., 179
Bellay, B., 201, 204–206, 213, 241, 261
Bellon, S., 260
Bennett, C., 192, 241, 265, 267
Benslimane, Y., 121, 127
Bercovier, M., 79
Berenbach, B., 250
Berger, C., 122
Berghel, H., 127, 144, 149
Berkley, C., 17, 58
Berkovich, E., 111
Bernstein, A. J., 99
Berry, D. M., 264
Berry, M. W., 13–14, 80
Best, C., 212, 245, 251
Beszedes, A., 194–195, 214
Beu, S. C., 35
Bevan, N., 222
Beynon, R. J., 35
Bialoalowv, M., 137–139
Biddle, R., 247
Biggerstaff, T. J., 252–253
Biggs, S. J., 177–178, 182
Bigham, J., 41
Binstock, A., 97
Birbaumer, N., 170–171, 173–174, 177, 182
Birch, G. E., 176
Bissling, H. R., 77, 111
Blagojevic, F., 9
Blaha, M., 205
Blankertz, B., 174
Block, B., 82
Boardman, R., 264
Bogdan, I., 35
Bogdanov, B., 28
Bonwick, J., 82
Bosch, J., 221–222
Bosworth, A., 238
Boulic, R., 177
Bowman, I. T., 209, 213–214, 237–238
Boyack, K. W., 12
Brainerd, J. G., 73
Brand, R., 212
Brandes, U., 239
Branner, A., 174, 177–179
Brehmer, S., 113
Brewer, E. A., 80
Briand, L. C., 270
Brichau, J., 216
Briggs, P., 10
Broberg, M., 176
Brown, A. W., 208
Brown, P., 202
Brown, R., 217–218
Brugger, T., 11, 48
Buck, I., 8–9
Buckley, J., 240–241, 264–266
Budgen, D., 270
Bufalari, S., 178
Buhler, J. D., 10
Bulbul, H. I., 136
Bull, M., 107
Bull, R. I., 192, 224, 229, 232, 241, 254–255, 265, 267
Burson, S., 212
Buss, E., 213, 229
Butler, B. S., 198

C

Camp II, D. G., 36
Cannon, W. R., 23, 36
Cantrill, B., 82
Caplan, A. H., 174, 177–179

Caprioli, R., 36
Carabalona, R., 172
Carmena, J. M., 177–178, 182
Carney, D. J., 200, 274
Carr, S. A., 36
Carrière, S. J., 207–208, 218
Carte, T. A., 198
Castro, S., 216
Cayirci, E., 155
Chabukswar, R., 87, 91, 108
Chalodhorn, R., 179
Chan, T., 80
Chang, P., 82
Chaparro, C. A., 36
Chapin, J. K., 177–178
Chapman, B., 73–74, 81, 111
Chappell, A., 17
Charles, J., 262
Chase, J., 17
Chase, M. P., 203
Chatterton, J., 18, 64
Chavarria-Miranda, D., 11, 46–48
Chen, I.-M. A., 23
Chen, J., 134–136
Chen, Y., 29, 36, 193–194, 203, 224, 246, 250
Cheney, J., 61
Cheng, L.-T., 242
Cherubini, A., 178
Chikofsky, E. J., 191
Choi, K., 180
Chong, J., 9
Christensen, S., 56
Chu, K., 23
Chute, C., 5
Cichocki, A., 180
Cieplak, P., 22
Cincotti, F., 178
Clark, C. R., 176
Clarke, C., 202, 216
Clauss, T. R., 30, 36
Clements, P., 200, 243–244, 265
Clothiaux, E. E., 53
Clowers, B. H., 38
Coca, D., 35
Cohen, J., 13
Comorski, D., 221
Confora, G., 263, 273
Congedo, M., 174
Conti, G., 42
Cook, K. A., 6
Copeland, J. A., 42
Copenhafer, M. A., 250–251
Coppit, D., 250–251, 256
Cordy, J. R., 215, 224, 234
Cormen, T., 112
Cortens, J. C., 32
Cova, G., 140
Covington, G. A., 9
Cowie, J. R., 12
Cox, A., 202, 216
Cox, D., 14
Craig, R., 31–32, 36
Crandall, R. W., 8
Cremer, K., 254
Crist, R. E., 177–178
Croes, H. J. P., 198
Cross II, J. H., 191
Cubranic, D., 201, 205, 217, 235–236, 242
Culler, D., 77, 87
Curran, E., 175
Cuthill, B., 207
Czarnecki, K., 249, 252–253
Czeranski, J., 205, 209

D

Daban, S., 172
Dalevi, D., 23
Dally, W. J., 86
Daly, D. S., 14
Damashek, M., 14
D'Ambros, M., 224, 242, 246, 256
D'Amico, A., 42
Dascal, S., 111
Davidson, G. S., 12

Davis, K. H., 212
Dayani-Fard, H., 203
Dean, J., 18
Dean, T. R., 215
D. L.sAngeles, D., 176
Demeyer, S., 192, 237–238, 260, 273
Demmel, J., 80, 87
Denker, M., 273
D. S.uza, J., 74
Devanbu, P. T., 232–234
Dewan, E. M., 172, 176
D'Hondt, T., 216
Dhong, S. H., 9
Di Penta, M., 263, 270, 273
Diamandis, E. P., 36
Diehl, S., 192
Dijkstra, E. W., 104
Dimitrov, D. F., 177–178
Distante, D., 235
Domeika, M., 113
Donato, J., 80
Donchin, E., 172
Dongarra, J., 80, 87
Donoghue, J. P., 174, 177–179
Doty, K., 56
Dourish, P., 207
Driesen, K., 217–218
Druker, B., 36
Dubchak, I., 23
Duca, K., 41
Ducasse, S., 194, 197, 200, 202–205, 207, 213–214, 227, 237–239, 242, 253–254, 256, 273
Dufour, B., 217–218
Dumais, S. T., 14
Dyba, T., 258, 264, 269

E

Eagan, R. C., 51, 56
Easterbrook, S., 206, 259–260, 270
Ebert, J., 194
Eckert, J. P., 73
Edlinger, G., 172
Edney, J., 121, 134–135, 143
Egyed, A. F., 250–251, 256
Ehrlich, K., 266
Eichberg, M., 216, 227, 248
Eiglsperger, M., 239
Eilam, E., 191
Eisen, J. A., 21–22
Eisenbarth, T., 205, 209
Eisenecker, U. W., 249, 252–253
Elbert, S. T., 6
Elden, C., 121
Elwood, M., 36
Eng, J. K., 31–32
Englebert, V., 192, 200, 207, 213
Erdos, K., 193
Estevez, J., 140
Estublier, J., 202, 228, 235, 244
Exton, C., 264–266

F

Fadrosh, D., 5, 29
Faloustsos, C., 11, 48
Farah, H., 267
Faria, A., 9
Farwell, L. A., 172
Fatourechi, M., 176
Favre, J., 202, 210, 214, 217, 228, 235, 244, 253
Favre, J.-M., 253–255
Feitelson, D. G., 81
Fellows, M. R., 177–178
Feng, L., 41
Fenton, N., 192
Fenyo, D., 32
Feo, J., 10–11, 48
Ferenc, R., 194–195, 203, 206, 209, 214, 228, 239, 260
Fernandes, K. J., 191
Fernando, R., 8
Ferrante, J., 80
Ferrez, P. W., 179, 182

Fetz, E. E., 177
Finkelstein, A., 206, 272
Finnigan, P. J., 194, 250
Fishman, D. A., 36
Fitzbibbon, S. P., 176
Fiutem, R., 204, 250
Flachs, B., 9
Flath, L., 8
Flor, H., 177
Flotzinger, D., 174
Folmer, E., 221–222
Fong, P., 205, 225, 259
Foote, H. P., 14
Forneris, C. A., 172–174
Fossi, M., 38, 40
Fracchia, F. D., 197, 223, 230, 267–268
France, R., 252
Frank, A., 29, 36
Frank, R., 8
Frazier, M., 5, 29
French, J. C., 248, 250
Friehs, G. M., 174, 177–179
Froehlich, J., 207
Fu, L., 200, 203, 240
Fulop, L. J., 206, 260
Fusaro, V. A., 36

G

Galan, F., 179, 182
Gall, H., 201, 211, 261
Gallagher, K., 206, 208
Gallis, H., 264
Gamez, D., 41
Gansner, D. R., 224, 250
Gansner, E. R., 218
Gantz, J. F., 5
Gao, Y., 41
Garlan, D., 243, 251
Garland, M., 9
Gatlin, C. L., 32
German, D. M., 192, 201, 205, 217, 235–236, 242, 274
Gerstner, W., 179
Gervais, G., 9
Ghanayim, N., 177
Gharachorloo, K., 104
Ghemawat, S., 18
Ghezzi, G., 211
Gibson, G., 11, 48
Gibson, T. D., 20
Gillette, M. A., 36
Girard, J., 200, 214, 237
Gîrba, T., 194, 204, 218–219, 239, 242, 253–254, 273
Glass, J. I., 5, 29
Glass, R. L., 234
Gochman, S., 87, 91, 108
Godfrey, M. W., 192, 206, 209, 213–214, 224, 229, 232, 237–238, 257
Gokhale, M., 13
Goldman, N. M., 250, 252, 256
Goldstine, H. H., 73
Goldwaithe, J., 177
Goncharova, I. I., 176
Gorton, I., 6, 17–18, 64
Govindaraju, N., 8
Govindaraju, N. K., 9
Gracio, D., 6
Grama, A., 86
Gramatica, F., 172
Gray, J., 6, 17
Grechkin, Y., 23
Green, T. R. G., 227, 267
Greenhill, D., 10
Gribschaw, H., 42
Griffin, P. R., 32, 113
Griswold, W. G., 201, 216
Grizzard, J., 42
Grudin, J., 226, 236
Guan, X. H., 41
Guéhéneuc, Y.-G., 201, 256, 261
Guerrero, E., 140
Guger, C., 172, 180
Guo, S., 41

Gupta, A., 86–87
Gutwenger, C., 250
Gyimóthy, T., 194–195, 206, 209, 214, 228, 239, 260

H

Hainaut, J., 192, 200, 207, 213
Hall, W., 268, 270
Halpern, A. L., 5, 21–22, 29
Hambrush, S., 77
Hammond, L., 74
Hamou-Lhadj, A., 192, 200–201, 203–204, 206–207, 209, 220, 240
H. A.Müller, 250
Hannay, J. E., 263–264
Hansen, O., 263–264
Harper, D., 10
Harris, D. R., 203
Harris, M., 8
Harrison, W., 236
Hartmann, J., 250
Hartwell, L., 36
Hassan, A. E., 191
Hassan, M., 10
Hatsopoulos, N. G., 177–178
Hattori, L., 264
Haupt, M., 216, 227, 248
Hegedus, P., 206
Heidelberg, A. B., 5, 29
Heidelberg, K. B., 21–22
Heinz, L., 227
Hellberg, C. S., 10
Helmreich, S. C., 12
Helms Tillery, S. I., 174, 177–178
Hennessy, J. L., 112
Henrard, J., 192, 200, 207, 213
Henriquez, C. S., 177–178
Henshaw, J., 213, 229
Heredia-Langner, A., 36
Herfurt, M., 139
Herlihy, M., 82
Herrera, F., 225–226, 233
Herrmann, C. S., 172
Hess, D. W., 10
Hey, T., 80
Hick, J.-M., 192, 200, 207, 213
Higgins, D., 17
Highfield, R., 5
Hildebrand, P. H., 53
Hill, H. H., 38
Himsolt, M., 239
Hind, M., 196
Hinterberger, T., 177, 182
Hirschfeld, R., 273
Hitt, B. A., 36
Hoare, C., 268
Hochberg, L. R., 174, 177–179
Hoffnagle, G., 270
Hofmeister, C., 193
Hofstee, H. P., 9
Hollebeek, T., 256
Holt, J., 113
Holt, R. C., 191, 193–195, 203, 206, 209, 213–214, 228, 237–239, 250, 259–260, 270
Holten, D., 250
Holzinger, A., 221
Holzner, C., 172
Hooper, D., 205, 225, 259
Hopkins, B., 137
Hopkins, K., 205, 225, 259
Horn, D. M., 35–36
Howell, K. E., 28
Huagang, X., 140
Huang, S., 231–232, 235–236, 250, 264, 270
Huang, W., 9
Hugenholtz, P., 23
Hupfer, S., 242
Hurley, C., 140
Husbands, P., 9

I

Ian, G., 58
Iivari, J., 228

Inverso, S. A., 172
Ivanova, N., 23
Iversen, I., 177
Ives, B., 132

J

Jackson, D., 196, 216
Jacob, A. C., 10, 13
Jacobs, J. M., 36
Jacobs, M., 41
Jaeger, E., 17, 58
Jahnke, J., 196, 230
Jaitly, N., 30, 35–36
Jajodia, S., 41
Jared, C., 58
Jarman, J. H., 36
Jaroszewski, L., 22
Jasperson, J. S., 198
Jazayeri, M., 233–234, 273
Jeffries, R., 264
Jerding, D., 206, 214
Jiang, M., 134–136
Jin, D., 210
John, B. E., 223
Johnson, E., 38
Johnson, J., 8–9
Johnson, K. L., 51, 53
Johnsrude, I., 175
Jones, C., 197
Jones, H., 9
Jones, M., 17, 58
Jorgensen, M., 258, 269
Jost, G., 73–74, 81, 111
Jünger, M., 250

K

Kafadar, K., 42
Kahan, S., 10
Kalapa, P., 41
Kalas, I., 194, 250
Kalcher, J., 174
Kamil, S., 9
Kaminski, P., 250
Kamp, M., 195, 237
Kampenes, V. B., 263–264
Kangas, L. J., 30
Kaplan, I., 11, 48
Karahasanovic, A., 263–264
Karipis, G., 86
Karp, R., 77
Kartz, M., 8
Kastner, C. M., 9
Kates, J., 223
Kazman, R., 200, 207–208, 218, 243–244, 265
Kehoe, K. E., 56
Kellens, A., 216
Keller, R. K., 201, 203, 205, 216, 222–223, 235, 237
Kelly, S., 252
Kennedy, K., 98–99
Kennedy, P. R., 177
Kern, A., 82
Kerr, S., 194, 250
Keutzer, K., 9
Khan, M. K., 201
Khokhlov, A., 10
Kienle, H. M., 189–274
Kim, H. K., 178, 181–182
Kim, J., 177
Kim, R., 9
Kimmel, J. R., 38
Kitchenham, B., 259–260
Kitzmiller, T., 212
Klein, K., 250
Klein, M., 200, 243
Klint, P., 256
Knight, J. C., 248, 250
Knight, R. T., 172
Knuth, D. E., 97, 265
Ko, A. J., 267, 273
Kohn, E. C., 36
Kohn, S., 11, 48
Kollias, P., 53
Kollmann, R., 229

Konecny, P., 10
Kongetira, P., 10
Konig, P., 177
Korn, J. L., 224, 250
Korzeniewski, F., 23
Koschke, R., 193–194, 197, 200, 205–206, 209, 214, 228, 237–239, 242–243, 260
Koskimies, K., 196
Kostov, A., 178
Kotchoubey, B., 177
Kouzes, R. T., 6
Kozyrakis, C., 82
Kraemer, E., 270
Kralik, J., 177
Kramer, J., 206, 272
Krasser, S., 42
Krausz, G., 172, 180
Krinke, J., 260
Krishnamurthy, B., 244–245, 248–249
Kruchten, P. B., 250, 268, 270
Krueger, C. W., 236, 243
Kruger, J., 8
Krusienski, D. J., 172, 174, 177, 179, 181
Kubler, A., 177
Kuhn, A., 195, 209, 253–254, 257, 273
Kuhn, T. S., 77
Kühne, T., 192
Kuipers, T., 200, 204, 250
Kullbach, B., 194
Kumar, A., 10, 87, 91, 108
Kumar, V., 86
Kupke, J., 250

L

L. B.ar, J., 36
Lafferty, F. W., 35–36
Lakhotia, A., 260
Lakshmi, 137
Lamarche, F., 174
Lämmel, R., 256
Lamport, L., 104
Lan, E. S., 201
Lancaster, J. M., 10
Lander, E. S., 21
Lang, S., 270
Lanza, M., 197, 202, 205, 207, 213–214, 224, 227, 242, 246, 250, 256, 264
Lanzagorta, M., 10
L. P.dus, M., 82
L. P.ant, D., 205
Larus, J., 74
Larvie, P., 264
Laubach, M., 177
Laurie, A., 138
Laurie, B., 138
Le Goff, F., 122
Le, T., 9
Leary, A., 128
Leback, B., 98
Lebedev, M. A., 177–178, 182
Lecuyer, A., 174
Lee, C., 42
Lee, E. A., 17, 110
Lee, M. S., 46
Leenstra, J., 9
Lefohn, A. E., 8
Leipert, S., 250
Leiserson, C., 112
Leng, M., 29, 36
Lethbridge, T. C., 192–193, 200–204, 206–207, 209, 214, 220, 224–226, 229–231, 233–234, 236, 240–241, 266–268
Leuthardt, E. C., 174, 177
Levary, R. R., 190
Levine, A. A., 9
Levine, P. J., 36
Lew, E., 179, 182
Lewis, G., 206
Lewis, T., 176
Li, J., 260
Li, W., 22
Li, Y., 36
Liao, J., 82
Liberty, J., 9
Liborg, N.-K., 263–264

Lienhard, A., 273
Liljegren, J., 56
Lin, M., 9
Lin, T. Y., 206, 208, 250
Linder, S., 36
Lintern, R., 250–251
Liotta, L. A., 36
Lipman, D. J., 21
Lipton, M. S., 30
Litoiu, M., 230–231, 244, 251, 254–255
Liu, L.-K., 9
Liu, P., 9, 41
Liu, Q., 9
Liu, T., 36
Liu, Y., 9, 134–136
Llett, D., 145
Lockwood, J. W., 9
Long, C. N., 56
Lotte, F., 174
Loveless, S., 176
Löwe, W., 208, 246–247
Lucca, G. A. D., 263, 270
Ludäscher, B., 17
Luebke, D., 8
Luke, E. P., 53
Lundberg, J., 208, 246–247
Lungu, M., 218–219, 224, 246, 250, 256
Lutteri, G., 250
Lykidis, A., 23

M

Maccari, A., 218, 222
MacCoss, M. J., 28
MacDonald, L. R., 174
Macduff, M. C., 51
Mach, T., 38
Madden, T. L., 21
Maggioni, S., 250
Malinger, I., 13
Malton, A. J., 215, 224, 229, 232
Manchoridis, S., 224, 250
Mandelblat, J., 87, 91, 108
Manfrediz, A., 5
Mani, D. R., 36
Mann, M., 28
Manning, G., 21–22
Manocha, D., 9
Marburger, A., 254
Marchette, D., 42
Marciani, M. G., 178
Marcus, A., 221
Marcus, M., 73
Markosian, L., 212, 215
Markowitz, R. S., 177–178
Markowitz, V. M., 23
Marowka, A., 73, 75, 79–80, 82, 88, 107–108, 111
Marquez, A., 11, 46–48
Marshall, M. S., 239
Martin, A., 247
Martin, J., 195, 233, 237
Marvak, E., 205
Maschhoff, K., 46–48
Masselon, C., 29, 36
Massick, S. M., 38
Mattia, D., 178
Mattson, T., 74
Mauchly, J. W., 73
Mavrommatis, K., 23
Maxim, M., 129, 131
Mayrhauser, A. V., 191
Mcarthur, J., 5
M. C.rd, R., 56
M. C.rmack, A. L., 31
M. C.llough, J., 135
M. D.nald, W. H., 28
M. F.rland, D. J., 170–179, 181–182
M. L.fferty, S. W., 35
M. M.llan, R., 137
M. Q.erry, D. L., 14
Mehlich, M., 201, 204
Mehner, K., 250
Mellinger, J., 183
Mendelson, A., 87, 91, 108
Mendling, J., 238–239
Mens, K., 201, 257, 261

Mens, T., 192, 260, 273
Merlo, E., 204, 250, 260
Merman, I., 239
Merritt, R., 82
Meyer, M., 218–219
Meyer, U., 13
Meyers, S., 210, 214
Mezini, M., 216, 227, 248
Michael, B., 9
Michaud, J., 197, 245
Michaud, J. W., 212–213
Miles, D., 98
Mili, R., 197
Millan, J. R., 174, 179, 182
Miller, C. S., 5, 29
Miller, M. A., 13, 53
Miller, W., 21
Mills, G. B., 36
Miner, L. A., 172, 174, 176
Minton, S., 5
Mitra, P. P., 177
Mock, S., 58
Mohammed, E., 82
Moise, D. L., 204–205, 209, 224
Monroe, M. E., 29–30, 35–36
Moonen, L., 193, 196, 257
Moore, A., 46
Moore, M. M., 177
Moore, R. J., 30
Moran, A., 236
Moran, D. W., 174, 177
Moran, K. P., 53
Morgan, S. R., 32
Morris, C., 236
Morris, E., 274
Mosier, J. N., 222
Moss, E., 82
Mourifio, J., 179
Moxon, K. A., 177–178
Mudur, S., 201
Muggleton, S. H., 5
Mukand, J. A., 174, 177–179
Muller, G., 180
Müller, H. A., 189–274
Muller, K.-R., 174
Muller, M., 270
Muller-Putz, G. R., 177, 180
Mumby, M., 29, 36
Murphy, G. C., 196–197, 201, 250
Murray, A. R., 250, 267
Musallam, S., 177
Mutzel, P., 250
Myers, B. A., 267, 273
Mylopolous, J., 194, 250
Mylopoulos, J., 203

N

Nanda, A. K., 9
Natsev, A., 9
Nawathe, U. G., 10
Neat, G. W., 172–174
Neidermeyer, E., 171
Neil, M., 192
Neuper, C., 180
Newbery, F. J., 218–219
Newcomb, P., 212, 215
Nickolls, J., 9
Nicolelis, M. A. L., 177–178, 182
Niemeyer, E., 87, 91, 108
Nieplocha, J., 11, 21, 46–48
Nierstrasz, O., 194, 204, 239, 242, 273
Nikolopoulos, D. S., 9
Ning, L., 41
Nishimoto, M. Y., 193–194, 203, 246
Noble, J., 247
Noel, S., 41
North, C., 41
North, S. C., 218
Norton, D., 98
Notkin, D., 196–197, 201, 250
Nuttin, M., 179, 182
Nuzman, J., 111

O

O'Brien, G. W., 14
O'Brien, L., 206, 208
O'Brien, M. P., 264–266
Ockerbloom, J., 251
Oda, Y., 28
O'Doherty, J. E., 177–178
Oehmen, C. S., 14, 21, 23, 26, 28, 36
Ogao, P., 205, 262–263, 270, 273
Oh, H., 9
Oivo, M., 272
Ojemann, J. G., 174, 177
Oliker, L., 9
Oltvai, Z. N., 6
Olukotun, K., 10, 74, 85, 113
Oman, P. W., 241
O'Neill, D., 107
Orebaugh, A., 131
Oriolo, G., 178
Orr, B., 56
Owen, A., 175
Owen, H. L., 42
Owens, J. D., 8
Ozel, M., 136

P

Pacione, M. J., 207
Padda, H., 201
Padki, A., 23
Palaniappan, K., 23
Palermo, S., 82
Paminski, L., 177–178
Panas, T., 208, 246–247, 259
Park, H., 10
Park, J., 209
Park, S., 181–182
Parnas, D. L., 212, 269
Parsons, S., 10
Pasa-Tolic, L., 29, 36
Patil, P. G., 177–178
Patrick, B., 86
Patterson, D. A., 77, 112, 235
Patterson, J., 242
Paul, S., 224
Paulson, L., 42
Pavese, A., 264
Payne, T., 231–232, 236
Pearce, R., 13
Peinador, J. F., 9
Penn, D. R.D., 174, 177–179
Penny, W., 175
Penta, M. D., 263, 270
Peppler, R. A., 56
Perel, S., 178, 181
Perelgut, S. G., 194, 250
Perlmouter, J., 177
Perrone, M., 9
Pesaran, B., 177
Petre, M., 227, 267
Petricoin, E. F., 36
Petrini, F., 9
Petritis, K., 30
Petyuk, V. A., 30, 36
Pevzner, P. A., 29, 36
Pezaris, J. S., 177
Pfurtscheller, G., 170–171, 173–174, 177, 180
Philip, J., 36
Philips, J., 179, 182
Piazza, R., 246
Pidgeon, C., 201, 204
Pike, W., 44
Pingry, S., 11, 48
Pllio, D., 129, 131
Ploedereder, E., 209, 214
Pnevmatikatos, D., 10
Polak, M., 178
Pollak, W., 274
Pope, K. J., 176
Powell, A. L., 248, 250
Poynter, K., 120
Pozo, V., 80
Prakash, A., 224
Principe, J. C., 178
Purcell, T., 8

Q

Qian, W. J., 29–30, 36
Qiang, G., 140

R

Raja, V., 191
Ram, S., 209
Ramamoorthy, C. V., 193–194, 203, 246
Rao, R. P. N., 179
Rayside, D., 251
Reinsel, F. D., 5
Reinsel, S., 5
Reiss, S. P., 213, 216–217, 222–224, 248, 250–251
Rekdal, A. C., 263–264
Remington, K., 21–22
Renggli, L., 253–254
Renkens, F., 179
Repo, M., 137–138
Reshotko, M., 82
Reubenstein, H., 246
Ricardo, R., 140
Ricca, F., 250
Richardson, S., 56
Riede, E., 11, 48
Riediger, V., 194
Riegelsberger, J., 264
Rifkin, S., 235
Rinard, M., 196, 216
Riva, C., 193, 214, 218, 222, 237–239, 250
Rivers, J., 35
Rivest, R., 112
Robbes, R., 224, 246, 256, 260
Robert, F., 270
Roberts, J., 87, 104
Roberts, S. J., 175, 246
Rockhill, B., 36
Rockwood, A. L., 35
Rodden, K., 264
Roger, D., 10
Rogers, E. M., 228, 234
Rogers, R., 140
Roland, D., 192, 200, 207, 213
Romine, C., 80
Roper, M., 207
Rosenberg, R., 10
Ross, K. A., 9
Ross, S., 242
Röthlisberger, D., 273
Rudolph, L., 81
Rugaber, S., 196, 206, 214, 230, 241, 252
Rumpe, B., 252
Rupp, R., 177, 180
Rusch, D. B., 5, 21–22, 29
Ryder, B. G., 192

S

Saha, B., 82
Sahani, M. S., 177
Sahay, A., 77
Saleh, M., 174, 177–179
Salton, G., 14
Salus, P. H., 244
Samuelson, P., 190
Sanders, C., 131
Sanlaville, R., 202, 228, 235, 244
Santos, E., 77
Santucci, D. M., 177–178
Saraiya, P., 41
Sarnacki, W. A., 174–175, 177–179, 181
Satish, N., 9
Saunders, C. S., 198
Scarpazza, D. P., 9
Schäfer, T., 216, 227, 248
Schaffer, A. A., 21
Schalk, G., 174, 177–178, 182–183
Schauer, R., 203, 237
Schauser, K. E., 77
Scherer, R., 177, 180
Scherrer, C., 44, 46–48
Schirrmeister, F., 113
Schlichting, W., 5
Schloerb, D. W., 178, 182
Schlogl, A., 174
Schmidt, G., 14

Schneider, H., 132
Schneider, K. A., 215
Schneier, B., 141
Schroder, M., 174
Schudel, G., 155
Schürr, A., 195, 203, 237–238
Schwartz, A. B., 174, 177–178, 181
Scotchmer, S., 190
Sechi, S., 28
Seffah, A., 201
Segal, J., 236
Sekhon, R. S., 53
Sellami, A., 270
Sellers, E. W., 172, 177
Senko, M. W., 35
Sensalire, M., 205, 262–263, 270, 273
Serruya, M. D., 174, 177–179
Sethi, R., 192, 244
Shah, A. R., 14, 20, 23, 26, 28
Shaked, Y., 138–139
Shalf, J., 9
Shaw, J. H., 36
Shaw, J. L., 35
Shaw, M., 242–243
Shemer, A., 87, 91, 108
Shen, Y., 29, 36
Shenoy, P., 179
Shi, X., 41
Shi, Y., 56
Shu, H., 29, 36
Sidak, J. G., 8
Sieber, J. E., 264
Siems, W. F., 38
Sillitti, A., 269
Sim, S. E., 206, 209, 211, 213, 220, 228, 237–239, 243, 258–262, 266, 270
Simard, P., 9
Simon, D., 205
Simone, C., 36
Singer, E., 181
Singer, J. P., 193, 222, 224–225, 229, 234, 263–264, 266–268
Singh, J. P., 87
Singhal, M., 20
Siozios, K., 9
Sipior, J. C., 149
Sjoberg, D. I. K., 258, 263–264, 269
Skadron, K., 9
Skillcorn, D., 77
Skramstad, T., 201
Sloughter, J. M., 14
Smith, D. B., 196, 230–231, 240–241, 244, 274
Smith, J. R., 9
Smith, R. D., 28–30, 35–36
Smith, S. L., 222
Sneed, H. M., 193, 203
Snir, M., 111
Soloway, E., 266
Sonntag, K. L., 56
Soudris, D., 9
Sounder, T. S., 224, 250
Sourdis, I., 10
Spalding, M. C., 178, 181
Srinivasan, M. A., 177–178, 181–182
Srivastava, R.-G., 36
Srivastava, S., 36
Stamatakis, A., 9
Stambaugh, C. R., 177
Stanley, M., 194, 250
Staron, M., 259
Stasko, J., 42
St-Denis, G., 203, 237
Stege, U., 273–274
Stein, C., 112
Steinberg, S. M., 36
Steiner, R., 197
Steinkraus, D., 9
Sterman, M. B., 174
Stevens, M., 264
Stirewalt, K., 252
Stirewalt, R., 270
Stokes, M., 175
Stone, R. K., 174

Storey, M.-A. D., 192, 196–197, 220–221, 223, 225, 230–232, 235, 241, 244–245, 250–251, 254–255, 261–262, 264–269
Storey, M. D., 193–194, 197, 201, 205, 217, 223, 225, 230, 235–236, 240, 242, 259, 262–263, 267–268
Strand, W., 5
Strittmatter, E. F., 30
Ströhlein, T., 14
Subramonian, R., 77
Sullivan, K. J., 197, 250–251, 256
Sulston, J. E., 21
Sun, D., 236
Sutter, H., 74
Sutton, G., 5, 21–22, 29
Swaminatha, T., 121
Sykacek, P., 175
Symantec, M., 40
Systä, T., 196
Szalay, A., 6, 17
Szeto, E., 23

T

Talia, D., 77
Tallis, M., 256
Tanner, S., 29, 36
Tao, J., 17
Tarkiainen, M., 194–195, 214
Tau, J. F., 73
Taub, E., 177
Taylor, D. A., 174, 177–178
Taylor, D. E., 10
Taylor, D. M., 178
Telea, A., 218, 246–247, 250, 262–263, 270, 273
Tentler, A., 172
Thalmann, D., 177
Thomas, J. J., 6
Thornton, F., 140
Tichelaar, S., 200, 202–203, 207, 209, 213–214, 237–239
Tichy, W. F., 264
Tilley, S. R., 196–197, 202, 212–213, 230–233, 235–236, 240, 244, 250, 264, 270
Toledo, S., 13
Toledo-Crow, R., 36
Toleman, M. A., 222, 227
Tolic, N., 35
Tolvanen, J.-P., 252
Toncheva, A., 5
Tonella, P., 204, 250
Tosi, C., 250
Tran, J. B., 214
Trevors, A., 224, 229, 232
Trudeau, C., 201, 204–206, 213, 241
Tydlitát, B., 9
Tynan, D., 143

U

Ullman, J. D., 192, 244
Ulmer, C., 13
Upputuri, B., 10

V

Vacca, J. R., 155
Vaidya, S., 8–9
Valadon, G., 122
Valiant, L. G., 77, 111
van den Brand, M., 257
van der Pas, R., 73–74, 81, 111
Van Der Spiegel, J., 73
van Deursen, A., 193, 200, 204, 250
Van, H., 80
Van Orden, S. L., 35
Vanacker, G., 179, 182
Vandenberg, J., 17
Vans, A. M., 191, 230, 267–268
Varbanescu, A. L., 9
Vaughan, T. M., 170–177
Velliste, M., 178, 181
Venter, J. C., 5, 29
Verhoef, C., 256

Verma, M., 36
Verschure, P. F., 177
Vidal, J. J., 171–172
Villa, O., 9
Villanueva, J., 36
Vinson, N., 193
Vishkin, U., 110–111
Vitzthum, F., 36
Voinea, L., 246–247, 250
von Eicken, T., 77
von Mayrhauser, A., 230, 267–268, 270
von Neumann, J., 72–73

W

Wagener, R., 56
Wakefield, J., 128
Walenstein, A., 221, 231, 236, 260, 263, 265, 267–268, 271
Wallace, A. K., 176
Wallnau, K. C., 200
Walsh, K. R., 132
Wang, C., 236
Wang, H., 236
Wang, L. C., 29, 36
Wang, Q., 217–218
Wang, W., 217–218
Ward, B. T., 149
Ward, R. K., 176
Warriner, L., 10
Washburn, M. P., 28
Wasserman, A. I., 244
Waterston, R. H., 21
Webb-Robertson, B. J. M., 14, 23, 26, 28, 36
Weber, A., 250
Wegman, E. J., 14, 42
Welsh, J., 222, 227
Wermelinger, M., 260, 273
Werner, G., 23
Wessberg, J., 177
West, D., 201
Westfechtel, B., 254
Wettel, R., 250
Whitehead, J., 274
Whitford, A. S., 178, 181
Whitham, E. M., 176
Whitley, K., 42
Whitney, M. J., 197
Whitney, P. D., 14
Widener, K. B., 51, 53
Wieringa, R., 236
Wile, D., 256
Williams, S., 9
Williamson, S. J., 5, 21–22, 29
Willoughby, J. O., 176
Wills, L. M., 230
Winter, A., 195, 203, 220, 237–238, 262
Wolpaw, J. R., 170–179, 181–182
Wolters, D. A., 28
Wong, A., 14
Wong, K., 193–194, 196, 199–200, 202–205, 207–209, 214, 221, 223–225, 230–232, 234, 236–240, 244, 250, 258–259, 262–263, 267–268
Wood, M., 207
Woods, S. G., 206–208
Wool, A., 138–139
Woolley, C., 8
Wrange, L., 206
Wu, C. C., 28
Wu, J., 238
Wu, X., 250
Wuyts, R., 201, 256–257, 261
Wynne, A. S., 17–18, 64

X

Xheneti, I., 5
Xu, Y., 30

Y

Yan, B., 30
Yang, C., 14
Yang, Y., 250, 260
Yarchi, I., 79
Yates I. I. J. R., 28, 31–32
Yeh, A. S., 203

Yelick, P., 9
Yen, K., 10
Yi, Y., 9
Yoo, A., 11, 48
Yoo, W. A., 13
Yooseph, S., 5, 21–22, 29
Young, I., 82
Yu, S., 29, 36
Yu, Y., 203

Z

Zabriskie, S., 44
Zacharski, R. A., 12
Zacksenhouse, M., 178
Zandi, E., 29, 36
Zanfei, S., 250
Zanoni, M., 250
Zelkowitz, M. V., 206–207, 244, 270
Zetter, K., 138, 140
Zhang, J., 21
Zhang, Z., 21
Zhao, I., 23
Zhao, Y., 17
Zheng, W., 198
Zhiying, Z., 232
Zhu, Q., 250
Zhuge, H., 41
Zimmer, J. S., 29, 36
Ziv, T., 32
Zou, F., 250
Zubarev, R. A., 35–36

Subject Index

A

ARM Climate Research Facility (ACRF), 49–50
Atmospheric radiation measurement (ARM) program
  ACRF, 49–50
  cloud radar spectra processing, 53, 55–56
  data-collection, 51
  data-flow architecture, 51–52
  data transfer and storage requirements, 53–54
  metadata and data provenance, 61
  QCRad, data quality analysis scheme
    correction process, 57
    data validation process, 58
    diffuse shortwave irradiance, 56–57
    surface radiation data, 56
  retrieval techniques., 52
  service-oriented scientific workflow, 58–61
    advantages, 59
    broadband heating rate profile (BBHRP) VAP, 58, 60
    ''drag-and-drop'' design, 58
    high-level workflow structure, 61
    MeDICi architecture and Kepler workflow engine interaction, 58–59
  value-added products, 61–62

B

Bitonic sort algorithm, 86
Bluetooth technology
  applications, 138
  improper implementation, 138–139
  objective, 137
  proof-of-concept viruses and tools, 139–140
Brain–computer interfaces (BCI)
  acquisition modules, 183
  alpha rhythm, 171
  BCI2000 standard, 182–183
  brain's complex reaction, 181
  communication and control, 170
  digital electronics, 182
  interchangeable signal-processing modules, 183
  kinematic control, 180–181
  movement control
    invasive BCI, 177–178
    kinematic control and goal selection, 176–177
    noninvasive BCI, 177
    two-and three-dimensional control, 178
  neuromuscular disorders, 170–171
  research and development
    electrocorticographic (ECoG) activity, 174
    EMG, 176
    facial muscles, 176
    Morse code message, 172, 176
    motor imagery, 175–176
    navigation and auditory imagery, 175
    sensorimotor rhythms (SMRs), 172–174
    signal classification algorithm, 174
    stepwise linear discriminant function, 172
    user and system adaptation, 175
    visual evoked potentials, 171
    ''zero-training'' method, 174

Brain–computer interfaces (BCI) (Continued)
robotic and prosthetic devices
embodied prosthetic control, 178
humanoid robot, 179
invasive and noninvasive recording, 180
machine-learning approach, 179
motor cortex, 178
RIKEN and TOYOTA collaboration, 180
soft actuators, 181–182
surface electrodes, 171

C

Cabir virus, 139–140
Commwarrior virus, 140

D

Data dependency
automatic parallelization, 98
loop-carried dependence, 99, 102
loop dependence, 99
loop distribution transformation, 99–100
loop fusion transformation, 101
types, 98
Data-intensive computing (DIC)
atmospheric radiation measurement (ARM) program
ACRF, 49–50
cloud radar spectra processing, 53, 55–56
data-collection, 51
data-flow architecture, 51–52
data transfer and storage requirements, 53–54
metadata and data provenance, 61
QCRad, data quality analysis, 56–58
retrieval techniques., 52
service-oriented scientific workflow, 58–61
value-added products, 61–62
biological sequence analysis
biological network visualization, 25–26
downstream analysis, 22
high-performance computing, 26–28
linear sequence, 21–22
metagenomics, 22
ScalaBLAST algorithm, 22–23
scientific impact, 23–24
data analytics
analytic focus, 15
data reduction., 15
digital imagery, 14
high-volume streaming data, 13
summarization, 15
vectors, 14
data-intensive analysis tools
CLIQUE code, 45–46
data-object-level analysis, 49
encryption, 43
feature-level analyses, 49
NUANCE code, 44–45
partial dimensions tree (PDTree) application, 46–48
scalable visualizations, 49
StatsView tool, 43–44
tactical and strategic work, 42
data-intensive cyber analytics
attack signatures, 41
cross-scale analysis, 42
data reduction, 40–41
definition, 6
digital content, 4–5
enterprise computing network, 39
evolutionary graphics, 42
hardware architectures
active-storage systems, 12
data-parallel streaming processors, 7
field-programmable gate arrays, 9–10
graphics processing units, 8–9
multithreaded systems, 10–12
regular and irregular applications, 7
solid-state storage, 13
high-throughput proteomics analysis
accurate mass and time (AMT) tag approach, 29–30
fragmentation spectra, 28

mass/charge measurement, 29
proteome measurement, 30
proteomics definition, 28
scientific impact, 36–37
sequence-based clustering, 29
smart instrument control, 34–36
tandem mass spectrometry, 31–34
human-in-the-loop discovery, 40
intrusion detection system, 41
inverse Hadamard transform
Hadamard transform, 38
ion mobility separation (IMS), 37–38
mass-to-charge ratio, 37
real-time processing, 39
time-of-flight (TOF) mass spectrometer, 38
sheer data volume, 5–6
software infrastructures
data processing pipelines, 15–18
MeDICi, 18–20
source and destination IP addresses, 40
visualization, 41–42
Design Maintenance System (DMS), 201

E

Electroencephalogram (EEG)
amplifiers, 182–183
digital electronics, 182
evoked potentials, 171
Morse code message transmission, 172
nonbrain activity, 176
noninvasive electrodes, 181
sensorimotor cortex, 180
sensorimotor rhythms (SMRs), 172–174, 176
Encryptions
applications, 157–158
illusions
digital certificate, 160, 162
router, 159
virtual public network (VPN), 160, 162
Web server, 160–161
single key/secret key, 155
two keys/public key, 156–157

H

Hawthorne effect, 264
High-throughput proteomics analysis
accurate mass and time (AMT) tag approach, 29–30
fragmentation spectra, 28
mass/charge measurement, 29
proteome measurement, 30
proteomics definition, 28
scientific impact, 36–37
sequence-based clustering, 29
smart instrument control
alignment algorithms, 35–36
database searching algorithm, 36
feature discovery algorithms, 35
high-performance data analysis pipeline, 34
high-performance DIC, MeDICi, 36
in silico analytical process, 34
THRASH algorithm, 36
tandem mass spectrometry
advanced model spectra, 32
analogous proteomic assessments, 31
computational reanalysis, 34
fault-tolerant access, 33
probabilistic process, 31
spectral libraries, 32–33
tryptic peptide candidates, 31–32

I

IEEE802 Standards
encryption, 124
login process, 124
signal spectrum, 125, 127
Web surfing, 125–126
IEEE802.11 standards, 165
Integrated microbial genome (IMG), 23
Internet service provider (ISP), 123

K

Kuhn's paradigm shift
  normal science phase, 77–78
  preparadigm phase, 77
  revolutionary science phase, 78

L

Lock-free programming paradigm, 82

M

Man-in-the-middle (MITM) attack, 152–154
Manycore programming
  ENIAC machine, 73
  human factor
    parallel programming, teaching, 111–112
    wish list, 112–113
  IAS machine, 73
  Intel Core 2 Duo CPU T8100, 83
  loop parallelization, 83–85
  missing parallel computation model
    DARPA High Productivity Computing Systems (HPCS), 78
    Kuhn's paradigm shift, 77–78
    parallel architecture, 77
    Parallel Computing Laboratory, 78
    PRAM model, 77
    Universal Parallel Computing Research Center (UPCRC), 78
    Von Neumann model, 76
  multicore processors, 74–76
  parallel computing, 73
  parallel hardware obstacles
    cache coherence problem, 88–91
    distributed counter, 96–97
    false-sharing problem, 91–93
    multiarchitecture problem, 85–88
    OpenMP 2.0, 93
    serial counting, 94
    shared counter protection, 95–97
  parallel revolution, 76
  parallel software issues
    data dependency, 98–102
    data race condition, 105–106
    locks and deadlocks, 107–110
    memory consistency model, 102–105
  performance-wall, 73–74
  portability problem
    auto-tuners, 80
    logical structure, 79
    OpenMP, 81
    parallel performance portability, 79
    performance, 78–79
    software portability, 78
    template/skeleton/sketching concept, 80
    tuning parameters, 80
  power-wall, 73–75
  scalability challenge, 81–82
Memory consistency model
  Dekker's algorithm, 104–105
  invisible interleaving scenario, 102
  OpenMP solution, 102
  relaxed memory consistency model, 104
  reordering scenario, 103–104
  sequential consistency model, 104
Model-driven visualization (MDV), 254–255
Multithreaded systems
  cache mechanisms, 10
  Cray MTA-2 and XMT, 11–12
  memory latency, 10

P

Package-oriented programming (POP), 256
Parallel hardware obstacles
  cache coherence problem
    cache-memory coherency management, 91
    directory-based coherency protocols, 90
    INTEL core Duo system, 91
    multilevel cache-memory hierarchy, 88
    nonuniform memory access (NUMA), 90
    snoopy coherency protocol, 89
    uniform memory access (UMA), 90
  distributed counter, 96–97

false-sharing problem, 91–93
multiarchitecture problem
bitonic sort algorithm, 86
common guideline framework, 85
contemporary multicore processors, 87
Dijkstra's algorithm, 86
Intel multicore processors, 87–89
OpenMP 2.0, 93
serial counting, 94
shared counter protection, 95–97
Parallel software issues
data dependency
automatic parallelization, 98
loop-carried dependence, 99, 102
loop dependence, 99
loop distribution transformation, 99–100
loop fusion transformation, 101
types, 98
data race condition, 105–106
locks and deadlocks
exception-aware mutex, 110
incorrect locking hierarchy, 109–110
Intel threading building block (TBB) locking overheads, 108
mutexes, 107
OpenMP locking overheads, 107
timer-attached mutex, 110
memory consistency model
Dekker's algorithm, 104–105
invisible interleaving scenario, 102
OpenMP solution, 102
relaxed memory consistency model, 104
reordering scenario, 103–104
sequential consistency model, 104
Partial dimensions tree (PDTree) application
all dimensions tree (ADTree) data structure, 46
anomaly detection, 48
Cray MTA-2, 46–47
lightweight user communication (LUC), 47–48
Lustre file system, 47
network traffic, 46
Threadstorm client, 48
Pervasive Parallelism Laboratory (PPL), 78
Public wireless networks
advantages, 148
attacks
bogus access point, 151
jamming, 151–152
man-in-the-middle (MITM) attack, 152–154
threats, 150
Internet surfing, 148
survey results, 149–150
war driving tools, 149
Wi-Fi network, 147

Q

QCRad, data quality analysis scheme
correction process, 57
data validation process, 58
diffuse shortwave irradiance, 56–57
surface radiation data, 56

R

Routers
bandwidth stealing
hacking, 129
initiation process, 130
operation, 128–129
consequences, 132–133
countermeasures
corporation responsibilities, 134–135
manufacturer responsibilities, 135–136
user responsibilities, 136–137
eavesdropping, 130–131
functions
IEEE802 Standards, 123–126
Internet service provider (ISP), 123
myths, 126–128

S

ScalaBLAST code, 23
Secure socket layer (SSL), 132, 160–161
Sensorimotor rhythms (SMRs), 172–174
Service-oriented architectures (SOAs), 211
Service-oriented scientific workflow
  advantages, 59
  broadband heating rate profile (BBHRP) VAP, 58, 60
  ''drag-and-drop'' design, 58
  high-level workflow structure, 61
  MeDICi architecture and Kepler workflow engine interaction, 58–59
Service set identifier (SSID), 135
Single key/secret key encryption, 155
Smart instrument control
  alignment algorithms, 35–36
  database searching algorithm, 36
  feature discovery algorithms, 35
  high-performance data analysis pipeline, 34
  high-performance DIC, MeDICi, 36
  in silico analytical process, 34
  THRASH algorithm, 36
Smart phones
  attacks, 141
  BlueBag security experiment, 137
  Bluetooth technology
    applications, 138
    improper implementation, 138–139
    objective, 137
    proof-of-concept viruses and tools, 139–140
  consequences, 142–143
  countermeasures
    corporation responsibilities, 146–147
    manufacturer responsibilities, 145–146
    user responsibilities, 147
  illusions, 140
  phone virus development, 143–144
Software infrastructures
  data processing pipelines
    blueprint, 16
    cloud computing, 17–18
    data capture and storage, 15
    data warehouse, 17
    downstream analytics, 16
  MeDICi
    architecture, 19–20
    data movement, 18
    MIF component, 19–20
    ProvenanceListener, 20
    workflow application, 18–19
Software reverse engineering tools
  adoptability
    academic reward structure., 235
    adoption factors, 232–235
    CASE tools, 228
    compatibility, 228–229
    complexity, 229
    diffusion of innovation, 228
    observability, 229
    SHriMP tool, 236
    tool adoption research, 230–232
    trialability, 229
  collaboration, 241–242
  components, 194
    analyzers, 196–197
    extractors, 195–196
    repository, 195
    visualizers, 197–198
  construction lens
    CodeCrawler software visualizer, 256
    component-based tool development, 247–252
    compositional and generative reuse, 243
    experimental software and toolkits (EST), 257
    idiosyncratic tools, 242
    model-driven tool development, 252–254
    package-oriented programming (POP), 256
    proof-of-concept implementation, 242
    tool architecture, 243–247
  customizability
    analyzers, 216
    data reverse engineering, 212

extensibility, 213
extractors, 215–216
mass-market software, 212
personalization, 220–221
repositories, 213–214
textual differencing algorithm, 220
visualizers, 216–220
definition, 190–191
diversity, 192
duplicate software, 191
elicitation and documentation, 240
evaluation lens
benchmark, 270
benchmarking, 259–260
case studies, 258–259
evaluation-driven tool building, 262–265
experiments, 259
feature analysis, 260–261
Kirchhoff's circuit law, 269
mutual fertilization, 271
SHriMP tool, 268
Storey's process iterations, 268–269
structured tool demonstration, 261–262
theory-grounded tool building, 265–268
treatment factors, 258
exchange formats
composability and formality, 237
domains, 239–240
file-based storage, 237
functional and nonfunctional requirements, 236
graph model, 237
incremental loading, 238
naming and querying, 238
neutrality and granularity, 237
popularity, 238
schema, 238–239
textual form, 237
version control, 237
exploration, 198–199
extraction, 192
interoperability
API, 209–210
CORUM frameworks, 208
definition, 206
schemas, 209
service-oriented architectures (SOAs), 211
tool integration, 207, 210
macro-level process, 193
micro-level process, 192–193
multiuser support, 241
quality attributes, 199–200
scalability
analysis, 204
computational performance and efficiency, 205–206
Design Maintenance System (DMS), 201
extractors, 203–204
repositories, 202–203
visualizers, 204–205
synthesis, 193
systems of systems (SoS), 274
trade-off analysis, 200
usability
characteristics, 221
CodeCrawler visualization tool, 227
definition, 221
importance of, 221–222
process-oriented usability, 224–225
product-oriented usability, 222–224
Sextant software exploration tool, 227
TkSee search tool, 225–226

T

Tandem mass spectrometry
advanced model spectra, 32
analogous proteomic assessments, 31
computational reanalysis, 34
fault-tolerant access, 33
probabilistic process, 31
spectral libraries, 32–33
tryptic peptide candidates, 31–32
Tool construction lens
CodeCrawler software visualizer, 256
component-based tool development (CBTD)

Tool construction lens (Continued)
adoptability, 252
application, 249–250
CIA tool, 248
customization, 251
FrameMaker, 248
interoperability, 251
scalability, 250–251
usability, 251
compositional and generative reuse, 243
experimental software and toolkits (EST), 257
idiosyncratic tools, 242
model-driven tool development (MDTD)
code generator, 252
domain-specific (modeling) language, 253
generative techniques and concepts, 252–253
REforDI tool, 254
Smalltalk code, 253–254
source and visualization meta-models, 254–255
package-oriented programming (POP), 256
proof-of-concept implementation, 242
tool architecture
control-driven approach, 244–245
data-driven integration framework, 244
presentation-driven integration, 246–247
quality attributes, 243
reference model, 244
ShriMP's architecture, 245
VizzAnalyzer, 246–247
Tool evaluation lens
benchmarking, 259–260
case studies, 258–259
discussion
benchmark, 270
Kirchhoff's circuit law, 269
mutual fertilization, 271
SHriMP tool, 268
Storey's process iterations, 268–269
evaluation-driven tool building
controlled experiments, 263–264
evaluation cycle, 262–263
first-and second-degree contact, 264
iterative activity, 262
paper-based prototypes, 264–265
experiments, 259
feature analysis, 260–261
structured tool demonstration, 261–262
theory-grounded tool building
cognitive theories, 266–267
HASTI, 267
program comprehension theories, 266–267
socio-cultural theory, 265
unsystematic theory generation, 268
Work Analysis with Synchronized Shadowing (WASS), 266
treatment factors, 258
Tool requirements lens
adoptability
academic reward structure., 235
adoption factors, 232–235
CASE tools, 228
compatibility, 228–229
complexity, 229
diffusion of innovation, 228
observability, 229
SHriMP tool, 236
tool adoption research, 230–232
trialability, 229
collaboration, 241–242
customizability
analyzers, 216
data reverse engineering, 212
extensibility, 213
extractors, 215–216
mass-market software, 212
personalization, 220–221
repositories, 213–214
textual differencing algorithm, 220
visualizers, 216–220
elicitation and documentation, 240
exchange formats
composability and formality, 237
domains, 239–240
file-based storage, 237

functional and nonfunctional requirements, 236
graph model, 237
incremental loading, 238
naming and querying, 238
neutrality and granularity, 237
popularity, 238
schema, 238–239
textual form, 237
version control, 237
interoperability
API, 209–210
CORUM frameworks, 208
definition, 206
schemas, 209
service-oriented architectures (SOAs), 211
tool integration, 207, 210
multiuser support, 241
quality attributes, 199–200
scalability
analysis, 204
computational performance and efficiency, 205–206
Design Maintenance System (DMS), 201
extractors, 203–204
repositories, 202–203
visualizers, 204–205
trade-off analysis, 200
usability
characteristics, 221
CodeCrawler visualization tool, 227
definition, 221
importance of, 221–222
process-oriented usability, 224–225
product-oriented usability, 222–224
Sextant software exploration tool, 227
TkSee search tool, 225–226
Two keys/public key encryption, 156–157

V

Virtual private network, 135
Virtual public network, 160, 162

W

War driving, 149–150
Web sites, 132
Wired equivalent privacy (WEP), 136
Wireless security
encryptions
applications, 157–158
illusions, 158–162
single key/secret key, 155
two keys/public key, 156–157
organization, 122
public wireless networks
advantages, 148
attacks, 150–153
Internet surfing, 148
survey results, 149–150
war driving tools, 149
Wi-Fi network, 147
routers
bandwidth stealing, 128–130
consequences, 132–133
countermeasures, 134–137
eavesdropping, 130–131
functions, 123–126
myths, 126–128
smart phones
attacks, 141
BlueBag security experiment, 137
Bluetooth technology, 137–140
consequences, 142–143
countermeasures, 145–147
illusions, 140
phone virus development, 143–144
user awareness, 121–122
Wi-Fi security survey, 164
Write-invalidate protocol, 89
Write-update protocol, 89

Contents of Volumes in This Series

Volume 42

Nonfunctional Requirements of Real-Time Systems
TEREZA G. KIRNER AND ALAN M. DAVIS
A Review of Software Inspections
ADAM PORTER, HARVEY SIY, AND LAWRENCE VOTTA
Advances in Software Reliability Engineering
JOHN D. MUSA AND WILLA EHRLICH
Network Interconnection and Protocol Conversion
MING T. LIU
A Universal Model of Legged Locomotion Gaits
S. T. VENKATARAMAN

Volume 43

Program Slicing
DAVID W. BINKLEY AND KEITH BRIAN GALLAGHER
Language Features for the Interconnection of Software Components
RENATE MOTSCHNIG-PITRIK AND ROLAND T. MITTERMEIR
Using Model Checking to Analyze Requirements and Designs
JOANNE ATLEE, MARSHA CHECHIK, AND JOHN GANNON
Information Technology and Productivity: A Review of the Literature
ERIK BRYNJOLFSSON AND SHINKYU YANG
The Complexity of Problems
WILLIAM GASARCH
3-D Computer Vision Using Structured Light: Design, Calibration, and Implementation Issues
FRED W. DEPIERO AND MOHAN M. TRIVEDI

Volume 44

Managing the Risks in Information Systems and Technology (IT)
ROBERT N. CHARETTE
Software Cost Estimation: A Review of Models, Process and Practice
FIONA WALKERDEN AND ROSS JEFFERY
Experimentation in Software Engineering
SHARI LAWRENCE PFLEEGER
Parallel Computer Construction Outside the United States
RALPH DUNCAN
Control of Information Distribution and Access
RALF HAUSER

Asynchronous Transfer Mode: An Engineering Network Standard for High Speed Communications
RONALD J. VETTER
Communication Complexity
EYAL KUSHILEVITZ

Volume 45

Control in Multi-threaded Information Systems
PABLO A. STRAUB AND CARLOS A. HURTADO
Parallelization of DOALL and DOACROSS Loops—a Survey
A. R. HURSON, JOFORD T. LIM, KRISHNA M. KAVI, AND BEN LEE
Programming Irregular Applications: Runtime Support, Compilation and Tools
JOEL SALTZ, GAGAN AGRAWAL, CHIALIN CHANG, RAJA DAS, GUY EDJLALI, PAUL HAVLAK, YUAN-SHIN HWANG, BONGKI MOON, RAVI PONNUSAMY, SHAMIK SHARMA, ALAN SUSSMAN, AND MUSTAFA UYSAL
Optimization Via Evolutionary Processes
SRILATA RAMAN AND L. M. PATNAIK
Software Reliability and Readiness Assessment Based on the Non-homogeneous Poisson Process
AMRIT L. GOEL AND KUNE-ZANG YANG
Computer-Supported Cooperative Work and Groupware
JONATHAN GRUDIN AND STEVEN E. POLTROCK
Technology and Schools
GLEN L. BULL

Volume 46

Software Process Appraisal and Improvement: Models and Standards
MARK C. PAULK
A Software Process Engineering Framework
JYRKI KONTIO
Gaining Business Value from IT Investments
PAMELA SIMMONS
Reliability Measurement, Analysis, and Improvement for Large Software Systems
JEFF TIAN
Role-Based Access Control
RAVI SANDHU
Multithreaded Systems
KRISHNA M. KAVI, BEN LEE, AND ALLI R. HURSON
Coordination Models and Language
GEORGE A. PAPADOPOULOS AND FARHAD ARBAB
Multidisciplinary Problem Solving Environments for Computational Science
ELIAS N. HOUSTIS, JOHN R. RICE, AND NAREN RAMAKRISHNAN

Volume 47

Natural Language Processing: A Human–Computer Interaction Perspective
BILL MANARIS

Cognitive Adaptive Computer Help (COACH): A Case Study
EDWIN J. SELKER
Cellular Automata Models of Self-replicating Systems
JAMES A. REGGIA, HUI-HSIEN CHOU, AND JASON D. LOHN
Ultrasound Visualization
THOMAS R. NELSON
Patterns and System Development
BRANDON GOLDFEDDER
High Performance Digital Video Servers: Storage and Retrieval of Compressed Scalable Video
SEUNGYUP PAEK AND SHIH-FU CHANG
Software Acquisition: The Custom/Package and Insource/Outsource Dimensions
PAUL NELSON, ABRAHAM SEIDMANN, AND WILLIAM RICHMOND

Volume 48

Architectures and Patterns for Developing High-Performance, Real-Time ORB Endsystems
DOUGLAS C. SCHMIDT, DAVID L. LEVINE, AND CHRIS CLEELAND
Heterogeneous Data Access in a Mobile Environment — Issues and Solutions
J. B. LIM AND A. R. HURSON
The World Wide Web
HAL BERGHEL AND DOUGLAS BLANK
Progress in Internet Security
RANDALL J. ATKINSON AND J. ERIC KLINKER
Digital Libraries: Social Issues and Technological Advances
HSINCHUN CHEN AND ANDREA L. HOUSTON
Architectures for Mobile Robot Control
JULIO K. ROSENBLATT AND JAMES A. HENDLER

Volume 49

A Survey of Current Paradigms in Machine Translation
BONNIE J. DORR, PAMELA W. JORDAN, AND JOHN W. BENOIT
Formality in Specification and Modeling: Developments in Software Engineering Practice
J. S. FITZGERALD
3-D Visualization of Software Structure
MATHEW L. STAPLES AND JAMES M. BIEMAN
Using Domain Models for System Testing
A. VON MAYRHAUSER AND R. MRAZ
Exception-Handling Design Patterns
WILLIAM G. BAIL
Managing Control Asynchrony on SIMD Machines—a Survey
NAEL B. ABU-GHAZALEH AND PHILIP A. WILSEY
A Taxonomy of Distributed Real-time Control Systems
J. R. ACRE, L. P. CLARE, AND S. SASTRY

Volume 50

Index Part I
Subject Index, Volumes 1–49

Volume 51

Index Part II
Author Index
Cumulative list of Titles
Table of Contents, Volumes 1–49

Volume 52

Eras of Business Computing
ALAN R. HEVNER AND DONALD J. BERNDT
Numerical Weather Prediction
FERDINAND BAER
Machine Translation
SERGEI NIRENBURG AND YORICK WILKS
The Games Computers (and People) Play
JONATHAN SCHAEFFER
From Single Word to Natural Dialogue
NEILS OLE BENSON AND LAILA DYBKJAER
Embedded Microprocessors: Evolution, Trends and Challenges
MANFRED SCHLETT

Volume 53

Shared-Memory Multiprocessing: Current State and Future Directions
PER STEUSTRÖM, ERIK HAGERSTEU, DAVID I. LITA, MARGARET MARTONOSI, AND MADAN VERNGOPAL
Shared Memory and Distributed Shared Memory Systems: A Survey
KRISHNA KAUI, HYONG-SHIK KIM, BEU LEE, AND A. R. HURSON
Resource-Aware Meta Computing
JEFFREY K. HOLLINGSWORTH, PETER J. KELCHER, AND KYUNG D. RYU
Knowledge Management
WILLIAM W. AGRESTI
A Methodology for Evaluating Predictive Metrics
JASRETT ROSENBERG
An Empirical Review of Software Process Assessments
KHALED EL EMAM AND DENNIS R. GOLDENSON
State of the Art in Electronic Payment Systems
N. ASOKAN, P. JANSON, M. STEIVES, AND M. WAIDNES
Defective Software: An Overview of Legal Remedies and Technical Measures Available to Consumers
COLLEEN KOTYK VOSSLER AND JEFFREY VOAS

Volume 54

An Overview of Components and Component-Based Development
ALAN W. BROWN
Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language
GUILHERME H. TRAVASSOS, FORREST SHULL, AND JEFFREY CARVER
Enterprise JavaBeans and Microsoft Transaction Server: Frameworks for Distributed Enterprise Components
AVRAHAM LEFF, JOHN PROKOPEK, JAMES T. RAYFIELD, AND IGNACIO SILVA-LEPE
Maintenance Process and Product Evaluation Using Reliability, Risk, and Test Metrics
NORMAN F. SCHNEIDEWIND
Computer Technology Changes and Purchasing Strategies
GERALD V. POST
Secure Outsourcing of Scientific Computations
MIKHAIL J. ATALLAH, K. N. PANTAZOPOULOS, JOHN R. RICE, AND EUGENE SPAFFORD

Volume 55

The Virtual University: A State of the Art
LINDA HARASIM
The Net, the Web and the Children
W. NEVILLE HOLMES
Source Selection and Ranking in the WebSemantics Architecture Using Quality of Data Metadata
GEORGE A. MIHAILA, LOUIQA RASCHID, AND MARIA-ESTER VIDAL
Mining Scientific Data
NAREN RAMAKRISHNAN AND ANANTH Y. GRAMA
History and Contributions of Theoretical Computer Science
JOHN E. SAVAGE, ALAN L. SALEM, AND CARL SMITH
Security Policies
ROSS ANDERSON, FRANK STAJANO, AND JONG-HYEON LEE
Transistors and 1C Design
YUAN TAUR

Volume 56

Software Evolution and the Staged Model of the Software Lifecycle
KEITH H. BENNETT, VACLAV T. RAJLICH, AND NORMAN WILDE
Embedded Software
EDWARD A. LEE
Empirical Studies of Quality Models in Object-Oriented Systems
LIONEL C. BRIAND AND JÜRGEN WÜST
Software Fault Prevention by Language Choice: Why C Is Not My Favorite Language
RICHARD J. FATEMAN
Quantum Computing and Communication
PAUL E. BLACK, D. RICHARD KUHN, AND CARL J. WILLIAMS
Exception Handling
PETER A. BUHR, ASHIF HARJI, AND W. Y. RUSSELL MOK

Breaking the Robustness Barrier: Recent Progress on the Design of the Robust Multimodal System
SHARON OVIATT
Using Data Mining to Discover the Preferences of Computer Criminals
DONALD E. BROWN AND LOUISE F. GUNDERSON

Volume 57

On the Nature and Importance of Archiving in the Digital Age
HELEN R. TIBBO
Preserving Digital Records and the Life Cycle of Information
SU-SHING CHEN
Managing Historical XML Data
SUDARSHAN S. CHAWATHE
Adding Compression to Next-Generation Text Retrieval Systems
NIVIO ZIVIANI AND EDLENO SILVA DE MOURA
Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java
LUTZ PRECHELT
Issues and Approaches for Developing Learner-Centered Technology
CHRIS QUINTANA, JOSEPH KRAJCIK, AND ELLIOT SOLOWAY
Personalizing Interactions with Information Systems
SAVERIO PERUGINI AND NAREN RAMAKRISHNAN

Volume 58

Software Development Productivity
KATRINA D. MAXWELL
Transformation-Oriented Programming: A Development Methodology for High Assurance Software
VICTOR L. WINTER, STEVE ROACH, AND GREG WICKSTROM
Bounded Model Checking
ARMIN BIERE, ALESSANDRO CIMATTI, EDMUND M. CLARKE, OFER STRICHMAN, AND YUNSHAN ZHU
Advances in GUI Testing
ATIF M. MEMON
Software Inspections
MARC ROPER, ALASTAIR DUNSMORE, AND MURRAY WOOD
Software Fault Tolerance Forestalls Crashes: To Err Is Human; To Forgive Is Fault Tolerant
LAWRENCE BERNSTEIN
Advances in the Provisions of System and Software Security—Thirty Years of Progress
RAYFORD B. VAUGHN

Volume 59

Collaborative Development Environments
GRADY BOOCH AND ALAN W. BROWN
Tool Support for Experience-Based Software Development Methodologies
SCOTT HENNINGER
Why New Software Processes Are Not Adopted
STAN RIFKIN

Impact Analysis in Software Evolution
MIKAEL LINDVALL
Coherence Protocols for Bus-Based and Scalable Multiprocessors, Internet, and Wireless Distributed Computing Environments: A Survey
JOHN SUSTERSIC AND ALI HURSON

Volume 60

Licensing and Certification of Software Professionals
DONALD J. BAGERT
Cognitive Hacking
GEORGE CYBENKO, ANNARITA GIANI, AND PAUL THOMPSON
The Digital Detective: An Introduction to Digital Forensics
WARREN HARRISON
Survivability: Synergizing Security and Reliability
CRISPIN COWAN
Smart Cards
KATHERINE M. SHELFER, CHRIS CORUM, J. DREW PROCACCINO, AND JOSEPH DIDIER
Shotgun Sequence Assembly
MIHAI POP
Advances in Large Vocabulary Continuous Speech Recognition
GEOFFREY ZWEIG AND MICHAEL PICHENY

Volume 61

Evaluating Software Architectures
ROSEANNE TESORIERO TVEDT, PATRICIA COSTA, AND MIKAEL LINDVALL
Efficient Architectural Design of High Performance Microprocessors
LIEVEN EECKHOUT AND KOEN DE BOSSCHERE
Security Issues and Solutions in Distributed Heterogeneous Mobile Database Systems
A. R. HURSON, J. PLOSKONKA, Y. JIAO, AND H. HARIDAS
Disruptive Technologies and Their Affect on Global Telecommunications
STAN MCCLELLAN, STEPHEN LOW, AND WAI-TIAN TAN
Ions, Atoms, and Bits: An Architectural Approach to Quantum Computing
DEAN COPSEY, MARK OSKIN, AND FREDERIC T. CHONG

Volume 62

An Introduction to Agile Methods
DAVID COHEN, MIKAEL LINDVALL, AND PATRICIA COSTA
The Timeboxing Process Model for Iterative Software Development
PANKAJ JALOTE, AVEEJEET PALIT, AND PRIYA KURIEN
A Survey of Empirical Results on Program Slicing
DAVID BINKLEY AND MARK HARMAN
Challenges in Design and Software Infrastructure for Ubiquitous Computing Applications
GURUDUTH BANAVAR AND ABRAHAM BERNSTEIN

Introduction to MBASE (Model-Based (System) Architecting and Software Engineering)
DAVID KLAPPHOLZ AND DANIEL PORT
Software Quality Estimation with Case-Based Reasoning
TAGHI M. KHOSHGOFTAAR AND NAEEM SELIYA
Data Management Technology for Decision Support Systems
SURAJIT CHAUDHURI, UMESHWAR DAYAL, AND VENKATESH GANTI

Volume 63

Techniques to Improve Performance Beyond Pipelining: Superpipelining, Superscalar, and VLIW
JEAN-LUC GAUDIOT, JUNG-YUP KANG, AND WON WOO RO
Networks on Chip (NoC): Interconnects of Next Generation Systems on Chip
THEOCHARIS THEOCHARIDES, GREGORY M. LINK, NARAYANAN VIJAYKRISHNAN, AND MARY JANE IRWIN
Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems
SHOUKAT ALI, TRACY D. BRAUN, HOWARD JAY SIEGEL, ANTHONY A. MACIEJEWSKI, NOAH BECK, LADISLAU BÖLÖNI, MUTHUCUMARU MAHESWARAN, ALBERT I. REUTHER, JAMES P. ROBERTSON, MITCHELL D. THEYS, AND BIN YAO
Power Analysis and Optimization Techniques for Energy Efficient Computer Systems
WISSAM CHEDID, CHANSU YU, AND BEN LEE
Flexible and Adaptive Services in Pervasive Computing
BYUNG Y. SUNG, MOHAN KUMAR, AND BEHROOZ SHIRAZI
Search and Retrieval of Compressed Text
AMAR MUKHERJEE, NAN ZHANG, TAO TAO, RAVI VIJAYA SATYA, AND WEIFENG SUN

Volume 64

Automatic Evaluation of Web Search Services
ABDUR CHOWDHURY
Web Services
SANG SHIN
A Protocol Layer Survey of Network Security
JOHN V. HARRISON AND HAL BERGHEL
E-Service: The Revenue Expansion Path to E-Commerce Profitability
ROLAND T. RUST, P. K. KANNAN, AND ANUPAMA D. RAMACHANDRAN
Pervasive Computing: A Vision to Realize
DEBASHIS SAHA
Open Source Software Development: *Structural Tension in the American Experiment*
COSKUN BAYRAK AND CHAD DAVIS
Disability and Technology: Building Barriers or Creating Opportunities?
PETER GREGOR, DAVID SLOAN, AND ALAN F. NEWELL

Volume 65

The State of Artificial Intelligence
ADRIAN A. HOPGOOD
Software Model Checking with SPIN
GERARD J. HOLZMANN

Early Cognitive Computer Vision
JAN-MARK GEUSEBROEK
Verification and Validation and Artificial Intelligence
TIM MENZIES AND CHARLES PECHEUR
Indexing, Learning and Content-Based Retrieval for Special Purpose Image Databases
MARK J. HUISKES AND ERIC J. PAUWELS
Defect Analysis: Basic Techniques for Management and Learning
DAVID N. CARD
Function Points
CHRISTOPHER J. LOKAN
The Role of Mathematics in Computer Science and Software Engineering Education
PETER B. HENDERSON

Volume 66

Calculating Software Process Improvement's Return on Investment
RINI VAN SOLINGEN AND DAVID F. RICO
Quality Problem in Software Measurement Data
PIERRE REBOURS AND TAGHI M. KHOSHGOFTAAR
Requirements Management for Dependable Software Systems
WILLIAM G. BAIL
Mechanics of Managing Software Risk
WILLIAM G. BAIL
The PERFECT Approach to Experience-Based Process Evolution
BRIAN A. NEJMEH AND WILLIAM E. RIDDLE
The Opportunities, Challenges, and Risks of High Performance Computing in Computational Science and Engineering
DOUGLASS E. POST, RICHARD P. KENDALL, AND ROBERT F. LUCAS

Volume 67

Broadcasting a Means to Disseminate Public Data in a Wireless Environment—Issues and Solutions
A. R. HURSON, Y. JIAO, AND B. A. SHIRAZI
Programming Models and Synchronization Techniques for Disconnected Business Applications
AVRAHAM LEFF AND JAMES T. RAYFIELD
Academic Electronic Journals: Past, Present, and Future
ANAT HOVAV AND PAUL GRAY
Web Testing for Reliability Improvement
JEFF TIAN AND LI MA
Wireless Insecurities
MICHAEL STHULTZ, JACOB UECKER, AND HAL BERGHEL
The State of the Art in Digital Forensics
DARIO FORTE

Volume 68

Exposing Phylogenetic Relationships by Genome Rearrangement
YING CHIH LIN AND CHUAN YI TANG
Models and Methods in Comparative Genomics
GUILLAUME BOURQUE AND LOUXIN ZHANG
Translocation Distance: Algorithms and Complexity
LUSHENG WANG
Computational Grand Challenges in Assembling the Tree of Life: Problems and Solutions
DAVID A. BADER, USMAN ROSHAN, AND ALEXANDROS STAMATAKIS
Local Structure Comparison of Proteins
JUN HUAN, JAN PRINS, AND WEI WANG
Peptide Identification via Tandem Mass Spectrometry
XUE WU, NATHAN EDWARDS, AND CHAU-WEN TSENG

Volume 69

The Architecture of Efficient Multi-Core Processors: A Holistic Approach
RAKESH KUMAR AND DEAN M. TULLSEN
Designing Computational Clusters for Performance and Power
KIRK W. CAMERON, RONG GE, AND XIZHOU FENG
Compiler-Assisted Leakage Energy Reduction for Cache Memories
WEI ZHANG
Mobile Games: Challenges and Opportunities
PAUL COULTON, WILL BAMFORD, FADI CHEHIMI, REUBEN EDWARDS, PAUL GILBERTSON, AND OMER RASHID
Free/Open Source Software Development: Recent Research Results and Methods
WALT SCACCHI

Volume 70

Designing Networked Handheld Devices to Enhance School Learning
JEREMY ROSCHELLE, CHARLES PATTON, AND DEBORAH TATAR
Interactive Explanatory and Descriptive Natural-Language Based Dialogue for Intelligent Information Filtering
JOHN ATKINSON AND ANITA FERREIRA
A Tour of Language Customization Concepts
COLIN ATKINSON AND THOMAS KÜHNE
Advances in Business Transformation Technologies
JUHNYOUNG LEE
Phish Phactors: Offensive and Defensive Strategies
HAL BERGHEL, JAMES CARPINTER, AND JU-YEON JO
Reflections on System Trustworthiness
PETER G. NEUMANN

Volume 71

Programming Nanotechnology: Learning from Nature
BOONSERM KAEWKAMNERDPONG, PETER J. BENTLEY, AND NAVNEET BHALLA
Nanobiotechnology: An Engineer's Foray into Biology
YI ZHAO AND XIN ZHANG
Toward Nanometer-Scale Sensing Systems: Natural and Artificial Noses as Models for Ultra-Small, Ultra-Dense Sensing Systems
BRIGITTE M. ROLFE
Simulation of Nanoscale Electronic Systems
UMBERTO RAVAIOLI
Identifying Nanotechnology in Society
CHARLES TAHAN
The Convergence of Nanotechnology, Policy, and Ethics
ERIK FISHER

Volume 72

DARPA's HPCS Program: History, Models, Tools, Languages
JACK DONGARRA, ROBERT GRAYBILL, WILLIAM HARROD, ROBERT LUCAS, EWING LUSK, PIOTR LUSZCZEK, JANICE MCMAHON, ALLAN SNAVELY, JEFFERY VETTER, KATHERINE YELICK, SADAF ALAM, ROY CAMPBELL, LAURA CARRINGTON, TZU-YI CHEN, OMID KHALILI, JEREMY MEREDITH, AND MUSTAFA TIKIR
Productivity in High-Performance Computing
THOMAS STERLING AND CHIRAG DEKATE
Performance Prediction and Ranking of Supercomputers
TZU-YI CHEN, OMID KHALILI, ROY L. CAMPBELL, JR., LAURA CARRINGTON, MUSTAFA M. TIKIR, AND ALLAN SNAVELY
Sampled Processor Simulation: A Survey
LIEVEN EECKHOUT
Distributed Sparse Matrices for Very High Level Languages
JOHN R. GILBERT, STEVE REINHARDT, AND VIRAL B. SHAH
Bibliographic Snapshots of High-Performance/High-Productivity Computing
MYRON GINSBERG

Volume 73

History of Computers, Electronic Commerce, and Agile Methods
DAVID F. RICO, HASAN H. SAYANI, AND RALPH F. FIELD
Testing with Software Designs
ALIREZA MAHDIAN AND ANNELIESE A. ANDREWS
Balancing Transparency, Efficiency, AND Security in Pervasive Systems
MARK WENSTROM, ELOISA BENTIVEGNA, AND ALI R. HURSON
Computing with RFID: Drivers, Technology and Implications
GEORGE ROUSSOS
Medical Robotics and Computer-Integrated Interventional Medicine
RUSSELL H. TAYLOR AND PETER KAZANZIDES

Volume 74

Data Hiding Tactics for Windows and Unix File Systems
HAL BERGHEL, DAVID HOELZER, AND MICHAEL STHULTZ
Multimedia and Sensor Security
ANNA HAĆ
Email Spam Filtering
ENRIQUE PUERTAS SANZ, JOSÉ MARÍA GÓMEZ HIDALGO, AND JOSÉ CARLOS CORTIZO PÉREZ
The Use of Simulation Techniques for Hybrid Software Cost Estimation and Risk Analysis
MICHAEL KLÄS, ADAM TRENDOWICZ, AXEL WICKENKAMP, JÜRGEN MÜNCH, NAHOMI KIKUCHI, AND YASUSHI ISHIGAI
An Environment for Conducting Families of Software Engineering Experiments
LORIN HOCHSTEIN, TAIGA NAKAMURA, FORREST SHULL, NICO ZAZWORKA, VICTOR R. BASILI, AND MARVIN V. ZELKOWITZ
Global Software Development: Origins, Practices, and Directions
JAMES J. CUSICK, ALPANA PRASAD, AND WILLIAM M. TEPFENHART

Volume 75

The UK HPC Integration Market: Commodity-Based Clusters
CHRISTINE A. KITCHEN AND MARTYN F. GUEST
Elements of High-Performance Reconfigurable Computing
TOM VANCOURT AND MARTIN C. HERBORDT
Models and Metrics for Energy-Efficient Computing
PARTHASARATHY RANGANATHAN, SUZANNE RIVOIRE, AND JUSTIN MOORE
The Emerging Landscape of Computer Performance Evaluation
JOANN M. PAUL, MWAFFAQ OTOOM, MARC SOMERS, SEAN PIEPER, AND MICHAEL J. SCHULTE
Advances in Web Testing
CYNTRICA EATON AND ATIF M. MEMON

Volume 76

Information Sharing and Social Computing: Why, What, and Where?
ODED NOV
Social Network Sites: Users and Uses
MIKE THELWALL
Highly Interactive Scalable Online Worlds
GRAHAM MORGAN
The Future of Social Web Sites: Sharing Data and Trusted Applications with Semantics
SHEILA KINSELLA, ALEXANDRE PASSANT, JOHN G. BRESLIN, STEFAN DECKER, AND AJIT JAOKAR
Semantic Web Services Architecture with Lightweight Descriptions of Services
TOMAS VITVAR, JACEK KOPECKY, JANA VISKOVA, ADRIANMOCAN, MICK KERRIGAN, AND DIETER FENSEL
Issues and Approaches for Web 2.0 Client Access to Enterprise Data
AVRAHAM LEFF AND JAMES T. RAYFIELD
Web Content Filtering
JOSÉMARÍA GÓMEZ HIDALGO, ENRIQUE PUERTAS SANZ, FRANCISCO CARRERO GARCÍA, AND MANUEL DE BUENAGA RODRÍGUEZ

Volume 77

Photo Fakery and Forensics
HANY FARID
Advances in Computer Displays
JASON LEIGH, ANDREW JOHNSON, AND LUC RENAMBOT
Playing with All Senses: Human–Computer Interface Devices for Games
JÖRN LOVISCACH
A Status Report on the P Versus NP Question
ERIC ALLENDER
Dynamically Typed Languages
LAURENCE TRATT
Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences
ADAM TRENDOWICZ AND JÜRGEN MÜNCH
Evaluating the Modifiability of Software Architectural Designs
M. OMOLADE SALIU, GÜNTHER RUHE, MIKAEL LINDVALL, AND CHRISTOPHER ACKERMANN
The Common Law and Its Impact on the Internet
ROBERT AALBERTS, DAVID HAMES, PERCY POON, AND PAUL D. THISTLE

Volume 78

Search Engine Optimization—Black and White Hat Approaches
ROSS A. MALAGA
Web Searching and Browsing: A Multilingual Perspective
WINGYAN CHUNG
Features for Content-Based Audio Retrieval
DALIBOR MITROVIĆ, MATTHIAS ZEPPELZAUER, AND CHRISTIAN BREITENEDER
Multimedia Services over Wireless Metropolitan Area Networks
KOSTAS PENTIKOUSIS, JARNO PINOLA, ESA PIRI, PEDRO NEVES, AND SUSANA SARGENTO
An Overview of Web Effort Estimation
EMILIA MENDES
Communication Media Selection for Remote Interaction of *Ad Hoc* Groups
FABIO CALEFATO AND FILIPPO LANUBILE