

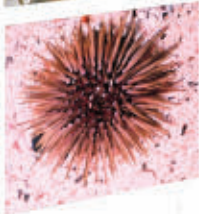
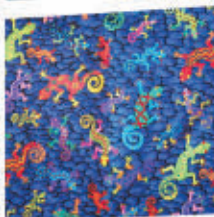
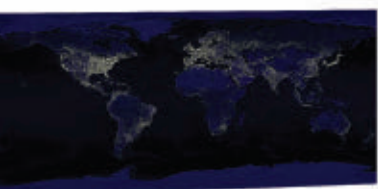
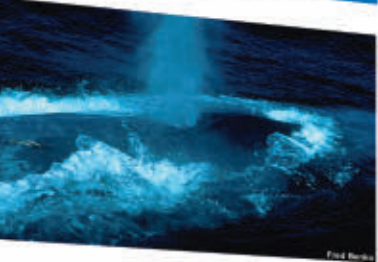
COMMUNICATIONS

CACM.ACM.ORG

OF THE

ACM

08/08 VOL.51 NO.8

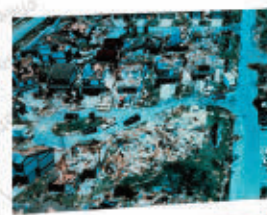


Designing Games with a Purpose

Computer Science and Game Theory

Scaling Games and Virtual Worlds

The Collaborative Organization of Knowledge



essays, {craft, art, science} of software, python, eclipse, agile development, onward!, {generative, functional} programming, .net, open source, concurrency, smalltalk, aspects, second life, ruby, service-orientation, objects, embedded, ultra large scale {model, test}-driven passion, fun!, agents, domain-specific use cases, movies, lightning talks,



systems, objective-c, development, c#, design patterns, languages, wiki, product-lines, java, refactoring, plop

DEFINE THE FUTURE OF SOFTWARE

www.oopsla.org/submit

CONFERENCE CHAIR

GAIL E. HARRIS

Instantiated Software Inc.
chair@oopsla.org

PROGRAM CHAIR

GREGOR KICZALES

University of British Columbia
papers@oopsla.org

ONWARD! CHAIR

DIRK RIEHLE

SAP Research
onward@oopsla.org

CALL FOR PAPERS

March 19, 2008

Due date for Research Program, Onward!, Development Program, Educators' Symposium, Essays and proposals for Tutorials, Panels, Workshops and DesignFest

July 2, 2008

Due date for Development Program Briefs, Doctoral Symposium and Student Volunteers



Association for
Computing Machinery

NASHVILLE CONVENTION CENTER, NASHVILLE, TN

October 19 - 23, 2008

Computer Science Springer References

SPRINGER
REFERENCE



Encyclopedia of Algorithms

Kao, Ming-Yang (Ed.)

The Encyclopedia of Algorithms will provide a comprehensive set of solutions to important algorithmic problems for

students and researchers interested in quickly locating useful information. The first edition of the reference will focus on high-impact solutions from the most recent decade; later editions will widen the scope of the work. This defining reference is published both in print and online.

Print

2008. Approx. 1220 p. 138 illus., Hardcover
ISBN 978-0-387-30770-1 ► **\$399.00**

eReference

ISBN 978-0-387-30162-4 ► **\$399.00**

Print + eReference

ISBN 978-0-387-36061-4 ► **\$499.00**



Encyclopedia of GIS

Shekhar, Shashi;
Xiong, Hui (Eds.)

The Encyclopedia of GIS provides a comprehensive and authoritative treatment of a dynamic

and rapidly expanding field, with entries alphabetically arranged for convenient, rapid access. The entries explain the key software, data sets, and processes used by geographers and computational scientists. All entries are contributed by world experts and peer-reviewed for accuracy and currency. The reference will be published as a print volume with abundant black and white art, and simultaneously as an XML online reference with hyperlinked citations and other interactive features.

Print

2008. XL, 1377 p. 723 illus., Hardcover
ISBN 978-0-387-30858-6 ► **\$399.00**

eReference

ISBN 978-0-387-35973-1 ► **\$399.00**

print + eReference

ISBN 978-0-387-35975-5 ► **\$499.00**



Encyclopedia of Multimedia

Furht, Borko (Ed.)

The Encyclopedia of Multimedia provides easily accessible coverage of the important concepts, issues and

technology trends in the field of multimedia technologies, systems, techniques, and applications. It is a comprehensive collection of more than 250 entries from hundreds of leading researchers and world experts. With over 1000 heavily-illustrated pages, the **Encyclopedia of Multimedia** presents concise overviews of all aspects of software, systems, web tools and hardware that enable video, audio and developing media to be shared and delivered electronically.

2006. XXIII, 989 p. 300 illus., Hardcover

ISBN 978-0-387-24395-5 ► **\$499.00**

Encyclopedia of Multimedia

Furht, Borko (Ed.)

Encyclopedia of Multimedia, Second Edition provides easy access to important concepts, issues and technology trends in the field of multimedia technologies, systems, techniques, and applications. Hundreds of leading researchers and world experts have contributed to this comprehensive collection of nearly 400 entries. Over 1200 heavily-illustrated pages (including 120+ new entries) present concise overviews of all aspects of software, systems, web tools and hardware that enable video, audio and developing media to be shared and delivered electronically.

Print

2nd ed. 2009. Approx. 1010 p. Hardcover
ISBN 978-0-387-74724-8 ► **\$449.00**

eReference

ISBN 978-0-387-78414-4 ► **\$449.00**

Print + eReference

ISBN 978-0-387-78415-1 ► **\$559.00**



Encyclopedia of Cryptography and Security

Tilborg, Henk C.A. van (Ed.)

This comprehensive encyclopedia provides easy access to information on all aspects of cryptog-

raphy and security. With an A-Z format of over 460 entries, 100+ international experts provide an accessible reference for those seeking entry into any aspect of the broad fields of cryptography and information security. Most entries in this preeminent work include useful literature references, providing more than 2500 references in total.

Print

2005. XII. 684 p. Hardcover
ISBN 978-0-387-23473-1 ► **\$349.00**

eReference

ISBN 978-0-387-23483-0 ► **\$299.00**

Print + eReference

ISBN 978-0-387-33557-5 ► **\$379.00**



Encyclopedia of Database Systems

Özsu, M. Tamer;
Liu, Ling (Eds.)

The multi-volume Encyclopedia of Database Systems provides easy

access to relevant information on all aspects of very large databases, data management, and database systems. Over 1,300 illustrated essays and definitional entries, organized alphabetically, present basic terminology, concepts, methods and data processing algorithms, key results to date, references to the literature, and cross-references to other entries and journal articles.

Print

2009. Approx. 4000 p. 60 illus.
In 4 volumes, not available separately
ISBN 978-0-387-35544-3 ► **\$1499.00**

eReference

ISBN 978-0-387-39940-9 ► **\$1499.00**

Print + eReference

ISBN 978-0-387-49616-0 ► **\$1899.00**

Departments

- 5 **President's Letter**
A New Beginning, A Fond Farewell
By Stuart I. Feldman
-
- 7 **Letters To The Editor**
Words Both Kind and Contrary
-
- 8 **CACM Online**
Small Changes Hint at Bigger Things
By David Roman

101 **Careers**

Last Byte

- 104 **Puzzled**
Delightful Graph Theory
By Peter Winkler

News

- 9 **Designing the Perfect Auction**
Distributed algorithmic mechanism design is a field at the intersection of computer science and economics.
By Hal R. Varian
-
- 12 **Access for All**
Accessible technologies are improving the lives of millions of physically impaired people around the world.
By Peggy Aycinena
-
- 15 **Challenging Poverty**
Information and communication technologies are an important component in the generation of wealth. How can they help reduce poverty?
By Sarah Underwood
-
- 18 **Remembering Jim**
Both melancholy and reverential, the Jim Gray Tribute at the University of California at Berkeley honored one of computer science's leading pioneers and visionaries.
By Michael Ross

Viewpoints

- 19 **The Profession of IT**
Voices of Computing
The choir of engineers, mathematicians, and scientists who make up the bulk of our field better represents computing than the solo voice of the programmer.
By Peter J. Denning
-
- 22 **From the Front Lines**
Software Development Amidst the Whiz of Silver Bullets
Software development organizations must accept the inevitability of silver-bullet solution proposals and devise strategies to defend against them.
By Alex E. Bell
-
- 25 **Education**
Paving the Way for Computational Thinking
Drawing on methods from diverse disciplines—including computer science, education, sociology, and psychology—to improve computing education.
By Mark Guzdial
-
- 28 **Viewpoint**
Envisioning the Future of Computing Research
Advances in computing have changed our lives—the Computing Community Consortium aims to help the research community continue that lineage.
By Ed Lazowska
-
- 31 **Interview**
Donald Knuth: A Life's Work Interrupted
In this second of a two-part interview by Edward Feigenbaum, we find Knuth, having completed three volumes of *The Art of Computer Programming*, drawn to creating a system to produce books digitally.
Edited by Len Shustek

Practice



- 38 **Scaling in Games and Virtual Worlds**
Online games and virtual worlds have familiar scaling requirements, but don't be fooled: Everything you know is wrong.
By Jim Waldo
-
- 45 **CTO Storage Roundtable**
Leaders in the storage world offer valuable advice for making more effective architecture and technology decisions.
By Mache Creeger, Moderator
-
- 52 **The Rise and Fall of CORBA**
There's a lot we can learn from CORBA's mistakes.
By Michi Henning

Contributed Articles



- 58 **Designing Games with a Purpose**
Data generated as a side effect of game play also solves computational problems and trains AI algorithms.
By Luis von Ahn and Laura Dabbish
-
- 68 **The Collaborative Organization of Knowledge**
Why Wikipedia's remarkable growth is sustainable.
By Diomidis Spinellis and Panagiotis Louridas

Review Articles



- 74 **Computer Science and Game Theory**
The most dramatic interaction between computer science and game theory may involve game-theory pragmatics.
By Yoav Shoham

Research Highlights

- 82 **Technical Perspective**
A Methodology for Evaluating Computer System Performance
By William Pugh
-
- 83 **Wake Up and Smell the Coffee: Evaluation Methodology for the 21st Century**
By Stephen M. Blackburn, Kathryn S. McKinley, Robin Garner, Chris Hoffmann, Asjad M. Khan, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot, B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann
-
- 90 **Technical Perspective**
Transactions are Tomorrow's Loads and Stores
By Nir Shavit
-
- 91 **Composable Memory Transactions**
By Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy

About the Cover: Ben Fry, doctoral graduate of the MIT Media Lab, took hundreds of actual images and responses from the gwap.com ESP Game and employed Processing from the gwap.com ESP Game and employed Processing to create the data visualization on this month's cover. Aside from being one of the original initiators of the open source Processing project, his book *Visualizing Data*, was recently published by O'Reilly.



Association for
Computing Machinery

Advancing Computing as a Science & Profession



COMMUNICATIONS OF THE ACM

A monthly publication of ACM Media

Communications of the ACM is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

Executive Director and CEO
John White
Deputy Executive Director and COO
Patricia Ryan
Director, Office of Information Systems
Wayne Graves
Director, Office of Financial Services
Russell Harris
Director, Office of Membership
Lillian Israel
Director, Office of Publications
Mark Mandelbaum
Director, Office of SIG Services
Donna Cappo

ACM COUNCIL
President
Wendy Hall
Vice-President
Alain Chenais
Secretary/Treasurer
Barbara Ryder
Past President
Stuart I. Feldman
Chair, SGB Board
Alexander Wolf
Co-Chairs, Publications Board
Ronald Boisvert, Holly Rushmeier
Members-at-Large
Carlo Ghezzi;
Anthony Joseph;
Mathai Joseph;
Kelly Lyons;
Bruce Maggs;
Mary Lou Soffa;
SGB Council Representatives
Norman Jouppi;
Robert A. Walker;
Jack Davidson

PUBLICATIONS BOARD
Co-Chairs
Ronald F. Boisvert and Holly Rushmeier
Board Members
Gul Agha; Michel Beaudouin-Lafon;
Jack Davidson; Carol Hutchins;
Ee-ping Lim; M. Tamer Ozsu; Vincent Shen;
Mary Lou Soffa; Ricardo Baeza-Yates

ACM U.S. Public Policy Office
Cameron Wilson, Director
1100 Seventeenth St., NW, Suite 507
Washington, DC 20036 USA
T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association
Chris Stephenson
Executive Director
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (800) 401-1799; F (541) 687-1840

Association for Computing Machinery (ACM)
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (212) 869-7440; F (212) 869-0481

STAFF

GROUP PUBLISHER
Scott E. Delman

Executive Editor
Diane Crawford
Managing Editor
Thomas E. Lambert
Senior Editor
Andrew Rosenbloom
Senior Editor/News
Jack Rosenberger
Web Editor
David Roman
Editorial Assistant
Zarina Strakhan
Rights and Permissions
Deborah Cotton

Art Director
Andrij Borys
Associate Art Director
Alicia Kubista
Assistant Art Director
Mia Angelica Balaquiot
Production Manager
Lynn D'Addesio
Director of Media Sales
Jonathan Just
Advertising Coordinator
Graciela Jacome
Marketing & Communications Manager
Brian Hebert
Public Relations Coordinator
Virginia Gold
Publications Assistant
Emily Eng

Columnists
Alok Aggarwal; Phillip G. Armour
Martin-Campbell-Kelly;
Michael Cusumano; Peter J. Denning;
Shane Greenstein; Mark Guzdial;
Peter Harsha; Leah Hoffmann;
Deborah Johnson; Mari Sako;
Pamela Samuelson; Gene Spafford;
Cameron Wilson

CONTACT POINTS
Copyright permission
permissions@acm.org
Calendar items
calendar@acm.org
Change of address
acmcoa@acm.org

WEB PRESENCE
<http://cacm.acm.org>

ADVERTISING

ACM ADVERTISING DEPARTMENT
2 Penn Plaza, Suite 701, New York, NY
10121-0701
T (212) 869-7440
F (212) 869-0481

Director of Media Sales
Jonathan M. Just
jonathan.just@acm.org

For the latest media kit—including rates—contact Graciela Jacome at jacome@acm.org

EDITORIAL BOARD

EDITOR-IN-CHIEF
Moshe Y. Vardi

NEWS Co-chairs
Marc Najork and Prabhakar Raghavan
Board Members
Brian Bershad; Hsiao-Wuen Hon;
Mei Kobayashi; Rajeev Rastogi;
Jeannette Wing

VIEWPOINTS Co-chairs
William Aspray and
Susanne E. Hambrusch
Board Members
Stefan Bechtold; Judith Bishop;
Peter van den Besselaar; Soumitra Dutta;
Peter Freeman; Seymour Goodman;
Shane Greenstein; Mark Guzdial;
Richard Heeks; Susan Landau;
Carlos Jose Pereira de Lucena;
Helen Nissenbaum; Beng Chin Ooi

PRACTICE Chair
Stephen Bourne
Board Members
Eric Allman; Charles Beeler;
David J. Brown; Bryan Cantrill;
Terry Coatta; Mark Compton;
Ben Fried; Pat Hanrahan;
Marshall Kirk McKusick;
George Neville-Neil

CONTRIBUTED ARTICLES Co-chairs
Al Aho and George Gottlob
Board Members
Yannis Bakos; Gilles Brassard; Peter
Buneman; Andrew Chien; Anja Feldman;
Blake Ives; Takeo Kanade; James Larus;
Igor Markov; Gail C. Murphy; Shree Nayar;
Lionel M. Ni; Sriram Rajamani; Avi Rubin;
Ron Shamir; Larry Snyder;
Wolfgang Wahlster; Andy Chi-Chih Yao;
Willy Zwaenepoel

RESEARCH HIGHLIGHTS Co-chairs
David A. Patterson and
Stuart J. Russell
Board Members
Martin Abadi; P. Anandan; Stuart K. Card;
Deborah Estrin; Stuart I. Feldman;
Shafi Goldwasser; Maurice Herlihy;
Norm Jouppi; Andrew B. Kahng; Linda
Petzold; Michael Reiter;
Mendel Rosenblum; Ronitt Rubinfeld;
David Salesin; Lawrence K. Saul;
Guy Steele, Jr.; Gerhard Weikum

WEB Co-chairs
Marti Hearst and James Landay
Board Members
Jason I. Hong; Jeff Johnson;
Wendy MacKay; Jian Wang

The Practice section of the CACM Editorial Board also serves as the Editorial Board of *ACM Queue*.

AUTHOR GUIDELINES
<http://cacm.acm.org/guidelines>

ACM Copyright Notice

Copyright © 2008 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

For other copying of articles that carry a code at the bottom of the first or last page or screen display, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center; www.copyright.com.

Subscriptions

Annual subscription cost is included in the society member dues of \$99.00 (for students, cost is included in \$42.00 dues); the nonmember annual subscription rate is \$100.00.

ACM Media Advertising Policy

Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current Advertising Rates can be found by visiting <http://cacm.acm.org/advertising> or by contacting ACM Media Sales at (212) 626-0654.

Single Copies

Single copies of *Communications of the ACM* are available for purchase. Please contact acmhelp@acm.org.

COMMUNICATIONS OF THE ACM

(ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

POSTMASTER

Please send address changes to *Communications of the ACM* 2 Penn Plaza, Suite 701 New York, NY 10121-0701 USA



Association for Computing Machinery

Printed in the U.S.A.



A New Beginning, A Fond Farewell

I am writing this column in my last month as President of ACM. It's been a great opportunity to support the Association's many successful programs and to expand and firmly establish new directions. Much has been accomplished, much remains to be done.

At this moment, you hold in your hands one of the biggest improvements—the newly renovated *Communications of the ACM*. From the outset of this ambitious project, our goal has been to make this a vibrant publication, a must-read and can't-wait-to-read for people everywhere who are excited by and depend on progress in computing. Readers need to know what is best and new in research, what is ripe enough to influence practice in a year or two, and what is happening in industry, government, and universities that affects the way we work.

The goal of the new *Communications* is to present a diverse collection of articles about the most interesting research in the field, as well as perspectives and reviews of hot topics, all written for knowledgeable and engaged computer scientists. It also brings articles about technology directions and problems that will interest practitioners and their managers. A new Practice section, aimed at computing professionals who develop, deploy, and enhance real systems, will leverage the success of the Association's respected *ACM Queue* magazine by having its editorial board serve as *Communications'* Practice board. In addition, you can now find news and analysis articles about people, organizations, funding, and directions in computing worldwide. Indeed, each section of the new

Communications has an editorial team assigned to select and shape its content. These teams unite leading voices from across the global computing field. I am eager to read upcoming issues.

ACM is striving to reach out to computer experts everywhere. Much of computing science, technology, and applications is location-independent, but the way people work is affected by where they live. Numerous activities are under way to make ACM more relevant to members outside the U.S. as well as to Americans with an increasingly global viewpoint: A growing number of ACM leaders—including elected officers, members of Council, as well as members of many ACM Boards and SIGs—are from outside the U.S. We have opened an office in Beijing to enable us to participate more fully in China. We also have advisory groups in China, India, and Europe to help ACM do more for our members and potential members in those areas.

In addition, we are working to address problems and concerns relating to our field. In many countries, university enrollment in the computing disciplines has been falling for years. Despite the centrality of information technologies to the economy and society, too many people think the bloom is off the rose. Yet new technological marvels arrive regularly because of the

fantastic work by computer scientists and engineers like you. In an effort to call more widespread attention to such marvels, ACM has undertaken a number of initiatives to address the image of the profession, including examining the role of policy, education, and diversity. We have planted the seeds; look for visible signs of growth in the coming years.

One way to increase visibility of the field, both within academia and in public, is through professional awards and press coverage. We have raised the financial levels of a number of ACM awards and instituted a new major prize—the ACM-Infosys Foundation Award—to recognize and honor great work in computer science. We have increased efforts to garner attention from the media and policy makers with our timely reports, boards, awards, and contributions from our excellent members.

As I noted in my opening remarks, much has been accomplished and much remains to be done. Thank you for allowing me this chance to serve the community and ACM. □

Stuart I. Feldman is vice president of engineering for Google, Inc., New York City.

introducing...

ACM's Newly Expanded Online Books & Courses Programs!

Helping Members Meet Today's Career Challenges

3,000 Online Courses from SkillSoft

The ACM Online Course Collection features **unlimited access to 3,000 online courses** from SkillSoft, a leading provider of e-learning solutions. This new collection of courses offers a host of valuable resources that will help to maximize your learning experience. Available on a wide range of information technology and business subjects, these courses are open to ACM Professional and Student Members.



SkillSoft courses offer a number of valuable features, including:

- **Job Aids**, tools and forms that complement and support course content
- **Skillbriefs**, condensed summaries of the instructional content of a course topic
- **Mentoring** via email, online chats, threaded discussions - 24/7
- **Exercises**, offering a thorough interactive practice session appropriate to the learning points covered previously in the course
- **Downloadable content** for easy and convenient access
- **Downloadable Certificate of Completion**

"The course Certificate of Completion is great to attach to job applications!"

ACM Professional Member

600 Online Books from Safari

The ACM Online Books Collection includes **unlimited access to 600 online books** from Safari® Books Online, featuring leading publishers including O'Reilly. Safari puts a complete IT and business e-reference library right on your desktop. Available to ACM Professional Members, Safari will help you zero in on exactly the information you need, right when you need it.



500 Online Books from Books24x7

All Professional and Student Members also have **unlimited access to 500 online books** from Books24x7®, in ACM's rotating collection of complete unabridged books on the hottest computing topics. This virtual library puts information at your fingertips. Search, bookmark, or read cover-to-cover. Your bookshelf allows for quick retrieval and bookmarks let you easily return to specific places in a book.



Association for
Computing Machinery

Advancing Computing as a Science & Profession

pd.acm.org
www.acm.org/join

Words Both Kind and Contrary

FOR THE FIRST time in many, many years I found *Communications* (July 2008) both friendly to the reader and useful. Over the years, the physical form and font of the issues were very unfriendly, let alone the content, which was quite rarified. Not this latest issue. The redesign makes the physical attributes—paper (non-glossy, yeah!), layout, and font very approachable. And the content is just great. Real concrete stuff to learn and use!

Makes me happy to have persevered as a member during the past 10 years!

Deepak Kenchamma, San Jose, CA

Thanks for the good work in revising *Communications* for readability and attractiveness. The new online format is very compelling as well. Readability for me is way up, and I like the catchy use of color.

Well done!

Alex Lancaster, Arlington, VA

I don't have time to adequately express my thoughts on the new design for *Communications*. About all I can say is, "Wow, I am impressed!" The content is so rich. Definitely a better magazine.

Thank you!

David Brown, Denver, PA

I received my print copy of the July *Communications* and wanted to let

you know how much I dislike the new three-column layout and reduced font size. I have been a member of ACM for over 25 years and have always enjoyed thumbing through every issue of *Communications*, reading articles here and there. Trying to read the new version was not enjoyable.

Has your design team forgotten that a lot of us ACM members are in our 50s, 60s, and 70s, and can't comfortably read small, wedged-together print?

Judy Walters, Naperville, IL

Catalan Is a Different Latin Language

I found it astonishing that the article "Web Searching in a Multilingual World" by Wingyan Chung (May 2008) included (in Table 2) the following claim: "Catalan (another version of Spanish) is widely used as well." A language spoken in the same country as another language is not necessarily "a version" of the other language. Catalan is, like French, Spanish, and Italian, a "different" Latin language, spoken not only in Spain but also in France and Andorra where it is the official language.

Alberto Gonzalez Tellez,

Valencia, Spain

Behavior Reinforcement is an Empirical Issue

Although the concepts were presented correctly in the section called "Feedback and learning from security-related decisions" in Ryan West's article "The Psychology of Security" (Apr. 2008), the definitions were incorrect. Reinforcements, both positive and negative, motivate behavior. The nagging messages regarding, say, Windows update availability can be seen as negative reinforcement. Users then update the operating system, and the message goes away. The same users would be more likely to update in the future if the message itself were, indeed, reinforcing.

Consequences that inhibit certain behavior are punishments, either positive (something "good" happens) or negative (something "bad" is re-

moved). Nothing should be assumed in advance to be reinforcing or punishing. Determining which is which is always an empirical issue. Only by studying how users actually respond to feedback from the interface can system developers truly understand how the interface affects their behavior. Otherwise, like a school principal wondering why students sent regularly to the "office" for "punishment" continue to misbehave, they will be left scratching their heads.

Timothy Dunnigan, San Diego, CA

Author's Response:

Thanks to Dunnigan for the correction. The distinction simply slipped my mind.

Ryan West, Round Rock, TX

The Price of Eternal Vigilance

Essentially the same quote was attributed to two different people, one in an article, the other in a column, in the same issue (Mar. 2008). The article "The Illusion of Security" by David Wright et al. ended with: "As Thomas Jefferson said, 'The price of freedom is eternal vigilance.'" The "Inside Risks" column by Xiaming Lu and George Leddin, Jr., began with: "When Wendell Phillips...told a Boston audience in 1852 that 'Eternal vigilance is the price of liberty'..."

A bit of Google-sleuthing found that although the quote is sometimes attributed to Jefferson, Tom Paine, and Patrick Henry, its first documented use was probably by Phillips (Bartleby.com's Dictionary of Quotations, www.bartleby.com/73/1073.html).

Further muddying the issue, wikiquote.org (en.wikiquote.org/wiki/Leonard_H._Courtney) claims the quote originated with Leonard Henry Courtney (1832–1918), a British baron, politician, and statistician.

Jeff Johnson, San Francisco

Communications welcomes your opinion. To submit a Letter to the Editor, please limit your comments to 500 words or less and send to letters@cacm.acm.org.

Coming Next Month in COMMUNICATIONS

Information Integration in the Enterprise

Design and Code Reviews in the Internet Age

Beyond Google: Automated Question Answering on the Web

How Do I Model State?

Plus the latest news on spectral graph theory, video encoding, and privacy technologies.

Small Changes Hint at Bigger Things

Hiding In Plain Sight

The redesign of *Communications* Web site is taking place largely out of sight, though a clue to the coming changes is on the cover of this very issue. It's sitting discretely under the name of the publication. It's the new URL for the new site, which will launch in early 2009: cacm.acm.org.

Calling All Authors

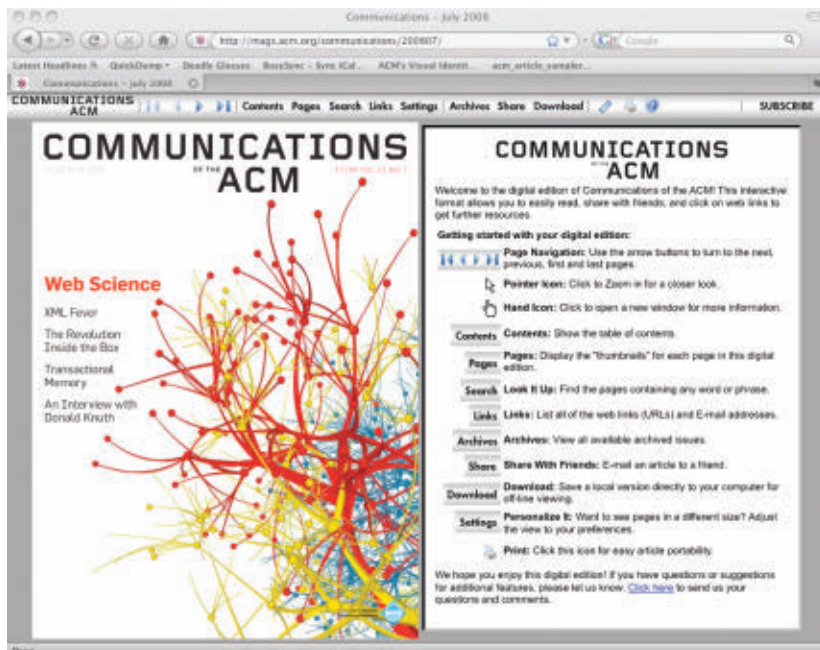
The dramatic changes to *Communications*—the magazine—call for an equally distinctive selection of top-notch articles. The Editorial Board of *Communications* welcomes your submissions, which can be delivered online at cacm.acm.org/submissions.

Rules to Write By

This new editorial model comes with a new set of guidelines for authors to follow. A comprehensive description of what's expected in a manuscript can be found in the Author Guidelines, available online at cacm.acm.org/guidelines.

Looking Good

You don't need Google to learn about authors of the nearly one million *Communications*' and other articles in ACM's Digital Library. The new Author Profile pages provide biographical information as well as usage statistics that help measure an author's impact. They also allow for community participation in the profiles. To investigate, click on an author's name from the full citation page of any article in the Digital Library; www.acm.org/dl.



Extra! Extra! Read All About It!

Each issue of *Communications* is available in a special Digital Edition, an electronic format that can be searched or downloaded. The latest Digital Edition, and an archive of past issues, is available online. The Digital Editions can also be viewed on a mobile phone, or via an RSS feed. Go to mag.acm.org/communications/.

ACM Member News

ACM PRESIDENTIAL AWARD WINNERS

The ACM Presidential Award was recently presented to four individuals. Steve Bourne was honored for his leadership in the creation of *ACM Queue* and the ACM Professions Board; Pat Ryan for her dedication to ACM and its many volunteers; Barbara Ryder for her efforts on behalf of numerous ACM committees, conferences, and councils; and Moshe Y. Vardi for his commitment to the ACM Job Migration Task Force and his new role as editor-in-chief of *Communications of the ACM*.

ACM-W INITIATIVE ON WOMEN AND MINORITIES

Tracy Camp, former co-chair of ACM's Committee on Women in Computing (ACM-W), is promoting a new project to attract women and other underrepresented groups to computing. This collaborative project, known as the Practices, Aggregation, Infrastructure, and Retrieval Service (PAIRS), is part of an ongoing ACM-W effort to develop a comprehensive collection of articles on women and minority groups in computing. So far, a collection of 135 resources, including research articles and teaching methods, are in the pipeline. For information about PAIRS, visit <http://www.colorado.edu/atlas/research/arc/pairs>.

ACM ELECTION RESULTS

Results of the recent General Election are:

PRESIDENT:	
Wendy Hall	4,783
J Strother Moore	3,918

VICE PRESIDENT:	
Alain Chesnais	4,907
Joseph A. Konstan	3,546

SECRETARY/TREASURER:	
Norman Jouppi	3,523
Barbara Ryder	4,965

MEMBERS AT LARGE:	
Carlo Ghezzi	6,172
Anthony Joseph	6,532
Mathai Joseph	4,789
Chuang Lin	3,399
Daniel Ling	4,482
Kelly Lyons	6,920
Mary Lou Soffa	6,324

Designing the Perfect Auction

Distributed algorithmic mechanism design is a field at the intersection of computer science and economics.

IN JANUARY, THE U.S. Federal Communications Commission (FCC) launched one of the biggest auctions in history, selling off what some in the wireless industry have called the “beachfront property” of the electromagnetic spectrum. Its auction of the 700MHz frequencies, the largest and most valuable slice of spectrum to come available in years, brought in more than \$19 billion.

With the auction, the FCC inaugurated the first use in a major spectrum auction of “package bidding,” in which bidders are allowed to bid either on individual state licenses or regional packages of licenses. Although package bidding makes a lot of sense—for example, some bidders might be interested in buying state A only if they can be guaranteed to also get state B—until now, the FCC had not offered an auction design that gave bidders enough flexibility while keeping combinatorial and computational complexity in check.

The auction is a high-profile example of distributed algorithmic mechanism design, a field that combines economics and algorithm design. Economic mechanism design is

concerned with how to design a market or market-like institution so that it will achieve a desired goal, such as allocating goods efficiently, maximizing profit, or achieving an equitable distribution. Mechanism design is, in a sense, the inverse of game theory: in game theory, one is given the rules of a game and the goal is to predict the outcome; in mechanism design, one

is given a set of desired outcomes and the goal is to design a game that will achieve them.

In 2007, Leo Hurwicz, Eric Maskin and Roger Myerson were awarded the Nobel Prize in Economics for their work in economic mechanism design. Their theoretical work underpins a host of practical applications, including eBay’s auctions, the auctions used by Google and Yahoo! to sell ad slots, the matching system used to pair medical residents and hospitals, the California electric power exchange, the rules governing trades on NASDAQ and other financial markets, and, of course, the FCC spectrum auctions.

As auctions have grown more complex and more economic transactions



occur online, computational considerations in mechanism design are taking on a growing significance. Distributed algorithmic mechanism design, or just algorithmic mechanism design, is emerging as a field in its own right.

“Mechanism design is one of the major intellectual interfaces between computer science and economics, as well as one of the most vibrant areas of economics,” says Christos Papadimitriou, a computer scientist at the University of California, Berkeley.

For Your Consideration

Broadly speaking, there are two main strands in the literature. The first involves bringing computational considerations to the economics mechanism design literature. The second involves bringing incentive considerations to the computer science literature.

As an example of the first issue, consider the recent spectrum auction by the Federal Communications Commission mentioned earlier. In this auction, the right to use spectrum in various locations was sold to mobile phone companies and other potential users. The valuation that a buyer places on spectrum in a particular location may depend strongly on whether or not it wins spectrum in other locations.

In theory, each buyer could assign a different value to each possible subset of the geographic locations being sold. How does one design an auction that will yield reasonable outcomes in such a “combinatorial auction”? In such auctions, it turns out that the so-called “winner determination problem” is, in general, NP-complete. However, researchers working in algorithmic mechanism design have discovered various approximation algorithms and special cases that allow for reasonably good solutions in practical examples.

As an example of the second issue, consider the famous “stable marriage problem” in which one wants to design an algorithm to match up men and women. Each man has a ranking over the women, and each woman has a ranking over the men. A stable assignment is one such that no couple would prefer to leave their current mates to form a new couple. Although this particular description may sound somewhat frivolous, there are much more serious examples, such as matching

A good starting point for studying distributed algorithmic mechanism design is a simple auction.

up hospitals and residents or organ donors and recipients.

It turns out that stable assignments always exist, and there are a number of algorithms that compute them. However, these algorithms assume that the participants are truthfully revealing their rankings. Do they actually have the appropriate incentives to do so? It turns out that some algorithms provide such incentives to men, and some provide such incentives to women, but there is no algorithm that provides incentives for both sides of the market to be truthful.

Ideas from economics can shed light on many computer science problems that arise from user interactions, such as computer viruses and spam, says Preston McAfee, a researcher at Yahoo! Research in Burbank, CA. “I think there’s a growing recognition that problems of bad behavior are incentive problems in the realm of game theory, rather than technological problems in the realm of traditional computer science,” he says.

Understanding the effect of incentives on how algorithms perform is “the latest and most momentous twist” on the question of computation’s limits, Papadimitriou says.

“With classical algorithms, you get your inputs and then compute away, and the answer comes out,” he says. “In this new context, you have to get your inputs by peering into the souls of selfish agents trying to promote themselves.”

A Simple Auction

A good starting point for studying distributed algorithmic mechanism design is a simple auction. A seller has one item to sell and n buyers have values

v_1, \dots, v_n for this item. The seller may have a reserve price r , which is the minimum price at which he is willing to sell the item. Typically there will also be a bid increment, the minimum amount by which a bid may be changed.

The goal is to design an online auction that will achieve some desired goals. There are many types of auctions that could be used. They include:

English auction. The seller starts at r and progressively raises the price by the bid increment until all but one of the buyers drops out. This is the most common form of auction.

Dutch auction. The seller starts at a high price and progressively lowers the price by the bid increment until a buyer shouts out “buy.” This sort of auction is used to sell flowers in the Netherlands.

First-price sealed bid. The buyers write down a bid and seal it in an envelope. The envelopes are opened and the item is awarded to the highest bidder at the price he or she bid. This form is commonly used for construction contracts.

Second-price sealed bid. The buyers write down a bid and seal it in an envelope. The envelopes are opened and the item is awarded to the highest bidder at the second-highest price. This auction was used by stamp collectors in the 19th century to sell stamps by mail.

It turns out there are some relationships among these auctions. For example, with fully rational players, the outcome of the English auction is the same, up to the bid increment, as the outcome of the second-price sealed bid auction. This is, perhaps, not so surprising upon reflection, as the English auction ends up awarding the item to the bidder who is willing to go the highest, but he or she only has to pay the bid of the second highest bidder plus a possible bid increment.

To make the argument slightly more precise observe that the payoff to a bidder with value v_1 is $v_1 - b_2$ where b_2 is the bid of the second-highest bidder. There are three cases to consider:

► $v_1 > b_2$. In this case bidder 1 wants to win. But the bidder can do so by reporting $b_1 = v_1$.

► $v_1 < b_2$. In this case bidder 1 wants to lose. But the bidder can do so by reporting $b_1 = v_1$.

► $v_1 = b_2$. In this case, bidder 1 is indifferent about winning or losing, so the bidder may as well report $b_1 = v_1$.

In each case it is optimal for bidder 1 to report his or her true value, regardless of what the bidder thinks other bidders will do. This is known as a *dominant* strategy in game theory. If everyone reports their true value, the item ends up being awarded to the bidder with the highest value, which is the efficient outcome in the sense of maximizing the value of the assignment.

The auction used by eBay is basically a form of second-price auction; the bidder who programs his or her bidding agent with the highest value wins, but only has to pay the second highest bid.

Note that in both of the examples mentioned—the 19th century stamp collectors and the eBay auction—the underlying motivation for adopting this auction form was communication costs. The stamp collectors did not want to mail bids back and forth and eBay buyers did not want to log on every time they wanted to change their bid.

Combinatorial Auctions

To continue with auctions, let us imagine a much more complex problem in which many items are to be sold. Let x represent an assignment of goods to bidders and let $v_a(x)$ represent agent a 's valuation—the agent's willingness to pay—for a given assignment. In principle, each agent may care not only about what he or she gets in the assignment, but also what everyone else gets. The seller does not know the bidders' valuation functions.

The auction design goal is to assign the items to the agents in a way that maximizes the sum of the individual valuations of the assignment.

Perhaps surprisingly, this mechanism design problem can be solved in much the same way as the single item auction. We simply ask each person to report their valuation functions. Next, we find the assignment that maximizes the sum of the reported valuations. The payment that agent a makes is the difference between the maximal value to the other agents if agent a is present and the maximal value if agent a is removed from the calculation. Roughly speaking, each agent has to pay the

cost that his or her presence imposes on the other agents.

To see how this generalizes the previous simple auction, note that in the simple auction the price that the highest bidder has to pay is the cost he or she imposed on the other agents; if the highest bidder weren't present, the second highest bidding agent would receive the item. This mechanism is known as the Vickrey-Clarke-Groves or VCG mechanism. It provides incentives to report true values for virtually any sort of problem. Of course, it also has flaws. For example, it does not typically generate the maximum amount of revenue for the seller.

Search Engine Ad Auctions

Google, MSN, and Yahoo! all use an auction to sell ad space on their search engines. Advertisers bid for positions on a search results page with the highest bidder receiving the most prominent position. The second highest bidder gets the second most prominent position and so on. Each advertiser pays a price per click based on the bid of the advertiser below him or her.

It turns out that there is no dominant strategy in this game when more than two positions being auctioned off. However, it is possible to find outcomes that are "stable" in the sense that no agent wants to change his or her bid, given the bids of the other advertisers.

Designing online auctions has been an evolutionary process, says Alvin Roth, an economist at Harvard University. "Google's design came out of some earlier designs, and getting the right design has been an important part of its success," he says. "It has helped create a market that didn't exist before."

Distributed algorithmic mechanism design offers an interesting theoretical framework for incorporating incentives into algorithmic design. It also offers exciting opportunities for interdisciplinary collaboration as well as being highly relevant to important practical problems, such as auctioning off the popular 700MHz frequencies. □

Hal R. Varian is the chief economist of Google.

Berkeley, CA-based science and technology writer Erica Klarreich provided additional reporting.

Information Technology

Video Search, Intel Style

Researchers at Intel labs in China and the U.S. are developing a video search technology that will enable users to search images in videos by person and object. Intel's video search technology divides videos on a frame-by-frame basis and uses image and face-recognition software to identify and categorize faces, voices, objects, locations, and movements. Next, the videos are reassembled to facilitate video search.

With Intel's video search technology, users will no longer have to fast forward through or watch an entire video, but can instantly cut to a particular scene or scenes. In addition to enabling users to instantly analyze videos, Intel's objective is to create a visual computing platform in which people can interact with a personal computer in a life-like, 3D environment.

Access For All

Accessible technologies are improving the lives of millions of physically impaired people around the world.

COUNTLESS PEOPLE INTERFACE with assistive technologies today either because they use them, develop them, or both. Some technologies have existed for years, but many more are rapidly emerging, motivated by fast-paced developments in science and engineering and by the allure of enormous potential markets.

Newly emerging technologies include mobile video phones for people who use sign language in combination with texting; enhanced optical character recognition and speech-synthesis tools that read books aloud; machine-learning algorithms and positioning sensors that enable a person in a wheelchair to better navigate an environment; improved speech recognition hardware for more accurately inputting verbal commands to a computer, wheelchair, or handheld device; and tools for designing more accessible Web sites.

More than 40 million Americans identify themselves as having a physical disability, of which 12 million use a computer and 17 million work full time, according to the U.S. Census Bureau. Globally, the United Nations estimates more than 700 million people have a physical disability. That figure is expected to grow due to improved health care and other factors that are increasing overall life expectancies. Factoring in the family members of these hundreds of millions, the market for assistive technologies encompasses several billion persons, and universities, companies, and governments are ramping up to meet the demand.

Profound changes are taking place in the assistive technology industry due to advances in compute power, signal processing, data compression, materials science, miniaturization, cognitive research, and the algorithms of artificial intelligence, along with a host of legal mandates and a growing awareness that full access to technology makes the world a happier, smarter, and more

productive place. Along with these technological advances, a 21st-century lexicon has emerged as well. People today talk about *accessibility technology*, rather than *assistive technology*.

Accessibility technology guru Richard E. Ladner, a professor in computer science and engineering at the University of Washington and winner of the 2008 A. Nico Habermann Award, notes that people don't want *assistance*; they want fair and equal access to computers, the Internet, consumer devices, and other aspects of 21st-century life no matter their preferences or needs. Ladner is also quick to point out that if

will be the ones to decide if any particular technology is part of that equation, so one of the biggest challenges is to find solutions that work and will also be adopted by a community."

Hence, a growing focus today is on *universal design*, making the human-machine interface fully configurable and responsive to everybody's needs with technology so customizable that it's accessible to all. That's the goal of today's dynamic, constantly evolving landscape of accessible technology research initiatives and commercial products.

"It's a Wild West out there," Ladner says. "In terms of the engineering



anyone expects to work in the field of accessibility technology, they must understand the accompanying terminology and the mindset.

There are no homogenous populations of accessibility technology users who can be lumped together by a common disability, Ladner says. There are only individuals who will evaluate the various accessibility tools made available and pick for themselves. "There are lots of examples of accessibility technology that were creative or inventive, but were never accepted," says Ladner. "People just want to live their lives, to succeed, and be happy. They

alone, accessible technology research is a wide-open field, with an infinite number of solutions."

Accessible Text

In Japan, a great deal of effort has gone into text captioning to make video broadcasting more accessible to people who are hearing impaired. At Kyoto University, various projects emphasize speech recognition and language processing for spoken text. At NHK Laboratory, part of Japan Broadcasting Corp., work focuses on real-time captioning in which a TV announcer's words are repeated by a speaker to produce a

higher, more stable rate of speech recognition for translation into text.

Speech-to-text conversion is far from perfect, however, as it is affected by factors such as audio devices, speaking style, and ambient noise. At the IBM Tokyo Research Laboratory, Takashi Saito, manager of the Accessibility Center, leads a group focused on correcting speech recognition errors. “This is tedious work,” says Saito. “First you listen to the audio to find errors in the text, then you delete the wrong characters, and then you input the correct characters. Our goal is to minimize the total time required for this process by simplifying the correction operations and minimizing the necessary keystrokes.”

Improving the quality of speech recognition is also playing a role in a proof-of-concept wheelchair at MIT. Finale Doshi, a graduate student in computer science, has designed a voice-activated wheelchair command system that uses machine learning to create and navigate a map of its environment. The wheelchair-bound person issues verbal commands to the guidance system to move from point to point on the map. High quality, easily trainable speech recognition devices that operate reliably are the key to implementing the wheelchair.

“People who use wheelchairs often have a lot of shaking, even people who don’t have several degenerative conditions,” says Doshi. “It takes far less mental concentration to maneuver a wheelchair if you can issue commands verbally rather than manually. This is a very active area of research at MIT.”

In Seattle, Ladner and his students in the Department of Computer Science and Engineering at the University of Washington have their own active areas of research. Their MobileASL project combines enhanced video compression with a cell phone configured as a video phone—the video lens is on the same side of the device as the phone’s screen, which has two panels, one of which displays the remote translator while the other panel displays the cell phone user—to provide more effective communication between people who sign and remote translators who provide American Sign Language (ASL) and text relay services.

Ladner insists the *raison d’être* for all accessibility technology is to

At MIT, a voice-activated wheelchair command system uses machine learning to navigate a map of its environment.

optimize people’s lives. “Accessible technology is about accepting, for instance, that people use sign language and making the phone adapt to their needs. It’s not about a prosthesis or replacing something that’s taken million of years to evolve. Not everybody wants a cochlear implant, which requires major surgery and can cause problems with balance.”

In conjunction with the Rochester Institute of Technology, Ladner’s group is also working to establish a DHH (deaf or hard of hearing) Cyber-Community between universities to increase enrollment of students who are deaf or hard of hearing in science, engineering, and mathematics from undergraduate levels through doctoral programs.

Also related to student access, Ladner’s group is developing a tool that translates textbooks, so a person who is blind can fully understand the content. “Between Braille and optical character recognition, words in textbooks are fairly accessible, but the figures are still difficult. We’re replacing figures with textures through an automated process using our Tactile Graphics Assistant,” he says.

Ladner’s students are also contributing to the growing worldwide effort to improve Internet accessibility. “Unfortunately, a lot of Web pages are not all that accessible for people who are blind or dyslexic,” he says. “Web designers use commercial development tools to make things look good, but don’t create a logical structure behind the page that’s navigable with a screen reader. Frequently, there’s also no alternative text inserted for figures.”

Computer Science

A Mind-‘Reading’ Computer

A pair of scientists at Carnegie Mellon University have created a computational model that can predict the brain activation patterns associated with concrete nouns, *Science* reports. Computer scientist Tom M. Mitchell and cognitive neuroscientist Marcel Just previously used functional magnetic resonance imaging (fMRI) to detect and pinpoint brain activity when a person thinks of a specific noun. With the fMRI data, the scientists created a computational model that enables a computer to determine what word a person is thinking of by analyzing brain scan data. In their latest research, Mitchell and Just used fMRI data to develop a computational model that can predict the brain activation patterns related to concrete nouns even if the computer did not possess fMRI data for a specific noun.

Mitchell and Just’s research could have applications in the study of autism, paranoid schizophrenia and other thought disorders, and semantic dementias such as Pick’s disease.

In response, Ladner's group has developed the WebInSight tool to infer the contents of a Web page and automatically insert alternative text. In addition, students Jeffrey P. Bingham and Craig M. Prince at the University of Washington are spearheading WebAnywhere, a low-cost, Web-based browser and self-voicing screen reader. (Commercial screen readers typically \$1,000.) WebAnywhere can also be used by developers evaluate the accessibility of their Web designs.

World Wide Access

When it comes to the World Wide Web, a host of accessibility technologies are in play or under consideration around the globe. The ACM Special Interest Group on Accessibility, SIGACCESS, has been showcasing novel ideas about computers and accessibility at their annual ASSETS conference for more than 10 years.

University of Manchester researcher Simon Harper is chair of this year's conference, which will be held in Halifax, Canada. "What we're doing is not just for a small subset of people, but for everybody," says Harper. "Global positioning systems, for instance, got started as speech recognition and positioning systems for people who are blind."

Harper is among those at the Human Centred Web Lab at the University of Manchester working to increase Internet accessibility. "Web designers make a lot of mistakes when they're designing Web sites, so we are studying how users interact with a dynamically updating page and where their attention is drawn to on the page," says Harper. "We believe by understanding

Like many accessibility researchers, Simon Harper believes accessibility starts with the design.

how users who are blind interact with a page, we can create novel methods of making obfuscated structures, information, and semantics more explicit in the design. We can help designers better understand which things on a page should be spoken and which should be more silent."

Like many accessible technology researchers, Harper believes accessibility starts with the design. "It would cost nothing and would be very easy to make a Web site from the outset that's supportive of accessible technology."

Vicki Hanson, chair of ACM SIGACCESS and a researcher at IBM's T.J. Watson Center in New York, agrees. She adds, however, that the decision to design for accessibility is more than just a matter of cleaning up the Internet—it's a matter of law.

"Section 508 of the Americans with Disabilities Act pertains to all businesses that the U.S. Government works with," Hanson says. "Every Web site for those businesses, and for all governmental agencies, has got to be designed for accessibility. Of course, if the costs

are too prohibitive, it won't happen for small businesses, so people in SIGACCESS are working to make accessibility features in software the standard, not something separate or different."

Cynthia Waddell, executive director of the International Center for Disability Resources on the Internet (ICDRI), says the move toward accessibility is a matter of international law. "When the U.S. government—the largest procurer of technology in the world—adopted Section 508 in 1998, people around the world started to realize they had better start to comply with best practices regarding accessibility. As of today, 126 countries have signed the 2006 U.N. Convention guaranteeing access to Information and Computer Technology (ICT) for people with disabilities. So much has happened over the last 10 years, it's almost unbelievable!"

ICDRI chair Mike Burks says accessible technology is about economics. "Some people maintain that pursuing accessible technology is too expensive, but people in the U.S. who have disabilities have an approximately 70% unemployment rate," says Burks. "That's a huge price for any society to pay for ICT not being accessible to all."

Simon Harper, however, says accessible technology is about choice. "Every one of us is bizarrely unique, and in the real world we do things in many different ways," he says. "There is no single solution to accessibility technologies. The solution is to have a whole menu of solutions from which each of us can pick and choose." ■

Peggy Aycinena is a freelance journalist based in Silicon Valley.

Internet

Dangerous Web Domains

Are some Web domains inherently more risky than others? According to software vendor McAfee's second annual *Mapping the Mal Web* report, the answer is a resounding "yes."

In its analysis of 9.9 million heavily visited Web sites in 265 different country and generic domains, McAfee found that the most dangerous Web domains

are those ending in ".hk" (Hong Kong), ".cn" (China) and ".info" (information). According to McAfee's report, almost one in five .hk sites (19.2%) are dangerous. Nearly 12% of both the .cn and .info domains were classified as dangerous.

A Web site with an .hk or .cn domain isn't necessarily located in Hong Kong or China; the owner of a domain name could

theoretically situate his or her business anywhere.

As for the world's most popular domain, ".com," slightly more than 5% of .com sites are deemed dangerous.

The three safest domains are ".gov" (government), with 0.05% classified as dangerous; ".jp" (Japan), with 0.1%; and ".au," (Australia) with 0.3%.

An unhealthy percentage of

Internet frauds involve the sale of fake pharmaceuticals.

"My advice about surfing behavior is that if you're really desperate for cheap Prozac and the pharmacy ends in .cn, don't do it. Just don't do it," said McAfee research analyst and report lead author Shane Keats in an interview with the Associated Press. "Find another place to get your Prozac."

Challenging Poverty

Information and communication technologies are an important component in the generation of wealth. How can they help reduce poverty?

AS THE DEBATE on whether poverty is best challenged by money or knowledge continues, efforts to improve individual lives and kick-start economies in developing countries are escalating. As in wealthy countries, where technology has transformed many lives, information and communication technologies (ICT) are part of development programs in poor countries. However, their application is very different and the implementation constraints can be overwhelming.

The World Bank, which cites its mission as “working for a world free of poverty,” is a supporter of ICT for development (ICT4D). The bank has a global ICT department with three organizational groups: one offers loans and assistance for ICT projects to developing world governments; another promotes sustainable private-sector investment in developing countries; and the last acts as an ICT think tank, bringing together and disseminating best practices.

This year, the World Bank will spend approximately \$7.3 billion on projects with an ICT component. Typical examples include an \$8 million grant to a private sector program in Bhutan that is establishing an IT park and a \$40 million loan to the government of Ghana for an e-Ghana project.

“We offer loans, grants, and technical assistance,” says Randeep Sudan, lead ICT policy specialist in the World Bank’s ICT department, “and we have a formal mechanism for deciding assistance strategies and working with governments to define projects and relationships. Inclusiveness and sustainability are key issues.”

The World Bank collaborates with many organizations, bringing together multidisciplinary teams including academics, consultants, anthropologists, computer scientists, and economists. Projects focusing on ICT consider how technology can impact poverty through

its application in areas such as education, health, agriculture, e-government, and public-sector reform.

With projects and people in place, the challenge is to overcome local constraints including a lack of ICT infrastructure, inadequate and unreliable power supplies, and a paucity of skilled, and sometimes literate, local people. Also, mind-sets need to be challenged and visionary plans created, particularly in developing countries that are lim-



ited by their own political or economic constraints.

Despite the difficulties of implementing technology, the World Bank sees ICT as an important element of transformation. “ICT has an impact in nearly every intervention we make to reduce poverty,” says Sudan. “It enhances employment, pushes up incomes, increases the employment of women, creates efficiency in government services, and reduces corruption.”

The European Commission also provides funds to sustain ICT4D initiatives and works in partnership with developing countries to build infrastructure. The Infrastructure Partnership with Africa, which the Commission supports,

is partially funding the EASSy submarine cable that will link the countries of East Africa to the rest of the world and is due to be in place before South Africa hosts the World Cup in 2010. As well as easing the lack of connectivity in Africa, the EASSy cable will provide lower communication costs than satellite systems.

Harry De Backer, a principal administrator working in the new technologies remit of the Commission’s European Development Fund, explains: “The EASSy cable will give Africa an opportunity to become part of the world economy through better communications, which will improve the export of locally produced products. EASSy will also provide backhauls into poorly connected areas of Africa, such as Kenya, Uganda, Burundi, Tanzania, and Rwanda. Ultimately, the backhaul will reach rural areas.”

With some 278 million mobile phones in Africa—one in three people has a mobile phone according to the GSM Association (GSMA), a global trade group of mobile phone operators—and GSMA operators poised to invest \$50 billion over the next five years, the prospect of creating a strong commercial environment is promising.

De Backer believes those living on just a few dollars a day will be included in the mobile phone community, stemming migration to congested cities and improving the lives of poor people through communication. One example of a mobile phone project is farmers who receive an SMS service telling them the consumer prices of vegetables. Armed with this information, the farmers can better negotiate prices with the middlemen who buy from the farmers and sell to consumers.

While connecting Africa is a major task, many smaller ICT projects are challenging poverty. Some have the potential to scale regionally, others could cross continents. Their proponents are experts with a desire to use ICT

to amplify what people in developing countries can do to improve their lives and eradicate poverty. Again, the task is Herculean, with the World Bank reporting that, despite a reduction in the proportion of people living in poverty in the developing world over the past 20 years from 40% to 20%, more than a billion people still struggle to survive on a dollar a day.

Improving Farmers' Lives

Successful ICT4D projects include eSagu, an IT-based personalized agricultural extension system that started in 2004 as a research project by the International Institute of Information Technology (IIIT) in Hyderabad, India and is funded by Media Lab Asia, a non-profit organization that carries out collaborative research in developing relevant and sustainable technologies, and culturally appropriate solutions, which will improve daily life.

In India, farming is the backbone of the economy, with two-thirds of the population living in rural areas and depending on agriculture for their income. However, the farming community faces numerous problems, including a lack of timely expert advice to help farmers be more productive and competitive.

eSagu ("Sagu" means "cultivation" in the Telugu language) aims to improve farm productivity by delivering farm-specific expert advice in an opportune manner to each farmer without the farmer needing to be literate or IT competent. The system is based on a team of agricultural experts at an

With HealthLine, women can become healthcare providers in rural villages that often have little or no health service provisions.

eSagu lab, usually in a city, supported by an agricultural information system. A small computer center, with a coordinator who is an educated and experienced farmer, covers a group of five or six villages. Every day, the coordinator visits farms to collect information and take photographs. A CD is then prepared and sent by parcel service—broadband is prohibitively expensive—to the main lab, where the experts analyze each farm's crop situation and prepare farm-specific advice. This is downloaded to the village eSagu center via a dial-up connection and the coordinator delivers the experts' advice to each farmer.

By closing the gap between agricultural research and practice, eSagu helps farmers improve efficiency and use pesticides and fertilizers effectively. An evaluation study showed that eSagu farms accumulated benefits worth about \$89 per acre.

IIIT professor P. Krishna Reddy, who has been involved in eSagu since

it started, suggests that the scalability of the system and its ability to be developed using existing infrastructure mean it could be expanded across rural India and replicated elsewhere.

"eSagu has been very successful. This year we will look at how it can be commercialized and improved further, still for the benefit of rural farmers," Reddy says.

At the Indian Institute of Technology (IIT) in Madras, professor Ashok Jhunjhunwala of the Department of Electrical Engineering, leads Tenet, a telecommunications and computer networking group that aims to bring not only telephony and Internet services to rural India, but also social improvement such as better education, agricultural development, and job creation. Jhunjhunwala also chairs a rural technology and business incubator with a mission to design, pilot, and nurture business ventures and a vision to facilitate inclusive technology and business development in rural areas.

"Everything is so different in rural areas compared to urban areas. The technology is different, connectivity is difficult and often only mobile, and the economics are different as there are a smaller number of people in a specific area with little ability to pay for services," explains Jhunjhunwala. "Each challenge is a huge learning experience and things you assume will work often don't."

While little connectivity in rural India 10 years ago meant there was no business case for commercial expansion, 60% to 70% of the rural population

Computer Science

Richard Karp Wins Kyoto Award

Richard Manning Karp was recently awarded the Kyoto Award in the category of Advanced Technology for his contributions to the theory of computational complexity, which he first developed in the early 1970s by establishing the theory of NP-completeness.

A professor of computer science and electrical engineering at the University of California, Berkeley, Karp has

had an enormous influence on the principles behind the analysis and design of algorithms used in numerous scientific disciplines.

Karp's NP-completeness theory increased the efficiency of problem solving by providing a standard method of measuring the computational complexity of combinatorial problems. His NP-completeness theory classifies problems by their degree of difficulty: Class P

represents problems for which polynomial-time algorithms of deterministic solutions exist and Class NP represents problems for which polynomial-time algorithms of non-deterministic solutions exist, including the sub-class NP-Complete, the most difficult-to-solve problems. By developing a standard methodology for this process, Karp significantly advanced the theory of computation and

algorithms that now support the field of computer science.

Karp is the recipient of the 1985 ACM A.M. Turing Award, the National Medal of Science, and the Benjamin Franklin Medal in Computer and Cognitive Science, among other awards. He will be presented with the Kyoto Award and a \$460,000 prize from the Inamori Foundation at an awards ceremony in Kyoto, Japan, in November.

is now connected, making it more feasible for telecom operators to move into rural India.

The business case around services based on connectivity remains weak, however, because the question of who will pay is unanswered. But Tenet and the IIT incubator are experimenting with a number of technology and application options, developing ideas that could scale to become commercial.

Jhunjhunwala forecasts that mobile communication will reach 97% of India's rural population in the next few years and that every village will have broadband in five or six years. However, he says, "we are also concerned about sustainable development and world issues such as climate change. Creating a better life for those in rural areas will challenge the poverty trap of moving to overcrowded urban areas and reduce climate damage."

Rural Healthcare

In Pakistan, ICT4D programs include a speech and language technology development research project led by Carnegie Mellon University and Aga Khan University, and initially funded by Microsoft's Digital Inclusion initiative. Called HealthLine, the project seeks to overcome a lack of healthcare information in rural areas by giving members of the healthcare community access to medical information. Healthcare workers, mostly village women chosen by the government for two months of basic training, use a toll-free number to call and ask questions of an automated health information system. The system overcomes literacy problems and barriers to information access, allowing the women to act as frontline healthcare providers in villages that often have little or no health service provision.

Jahanzeb Sherwani, an undergraduate from Lahore and a doctoral student at Carnegie Mellon, is working on the project in Karachi, talking to healthcare providers about their needs and considering how technology can be adapted for populations with a low level of literacy. The system is being tested and, if it is successful, could be scaled to cover the 100,000 rural healthcare workers in Pakistan. The economics of the system are good as health workers need only access to a phone and the health information is held on a PC server in Karachi.

Roni Rosenfeld, professor of computer science at Carnegie Mellon, hopes the Pakistani government will fund a large-scale version of the project, but also envisions a business model that requires people to pay a small fee for information they want, making the project self-sustaining if it is not government-funded.

Are such projects sustainable? "Absolutely," says Rosenfeld. "Although it is hard to predict sustainability for any one ICT4D project, overall sustainable projects are sure to emerge. We need expertise in IT, economics, social policy, different cultures, and business, and we need to try out as many ideas and solutions as possible. Some will fail, but some will succeed."

It is not just academic projects that are reaching the poorest people on the planet. Commercial companies are also playing a part. While cynics suggest their interest is in cornering emerging markets, corporations such as Microsoft take a more balanced view. Kentaro Toyama, a leader in Microsoft's Technology for Emerging Markets group at Microsoft Research India, acknowledges the business potential of new markets, but also points to the company's responsibility to help people get the most out of computers, particularly in places that have previously lacked access to technology.

In terms of ICT4D projects, Microsoft runs many, funding research budgets and collaborating with development partners such as the World Bank. Its projects include Digital Green, which disseminates agricultural education to small farmers through digital video, and text-free user interfaces, which allow nonliterate groups to access computers.

While the answer to the question of whether the end of poverty will be achieved by money or knowledge is probably both, Toyama adds the need for human interest. "The problems of developing countries are huge and dire," he says. "We have to do as much as we can to help by harnessing the energy of people in developed countries. ICT4D is sustainable and can be successful as long as it attracts human interest." ■

Sarah Underwood writes about computing and technology from Teddington, UK.

Patent Applications

Patent Filings Increase in China

More patent applications were filed in China than any other country last year, according to China's State Intellectual Property Office, which received 694,000 applications in 2007. The U.S. had the second most applications, with 484,955, followed by Japan with 443,150.

Three types of patents are granted in China: invention patents, which are valid for 20 years from the date of filing, and utility patents and design patents, both of which are valid for 10 years. In terms of invention patents, China is ranked third in the world, behind the U.S. and Japan. If China's number of patent applications continues at its current rate, it will lead the world in invention patent applications by 2012.

Approximately one-third of the invention patent applications filed in China are made by foreign businesses, "which clearly suggests that filing in China has become an intrinsic part of most multinational company's [intellectual property] strategies," according to Evaluateserve, a market and business research company.

Remembering Jim

Both melancholy and reverential, the Jim Gray Tribute at the University of California at Berkeley honored one of computer science's leading pioneers and visionaries.

ON MAY 31, almost 750 colleagues, friends, and family members gathered at the University of California at Berkeley campus for a day-long tribute to Jim Gray, who disappeared at sea while sailing his 40-foot sailboat off the coast of San Francisco on Jan. 28, 2007.

For nearly four decades, Gray worked for some of the computer industry's largest companies including Digital Equipment, IBM, Microsoft, and Tandem. His intellect and technical achievements in database and transaction processing are legendary. He won the ACM A.M. Turing Award in 1998 for his description of the basic requirements for transactions as well as his research in locking, concurrency, and fault tolerance. He initiated the use of performance benchmarks for a wide variety of transaction environments. And in recent years he stimulated the creation of massive distributed scientific databases that are reshaping the fields of astronomy and oceanography, and are laying the groundwork for eScience, a new method of research.

Even more impressive than his professional achievements is the fact that hundreds of Gray's colleagues considered themselves to his best friend, so genuine and deep was his interest in each of them. Accordingly, the tribute's audience included graduate students and new employees, astronomers and geologists, department chairs and CEOs, and several generations of computer scientists and database experts from academia and industry. A handful of attendees traveled from as far away as Europe and Asia solely for the event.

Gray entered the University of California at Berkeley in 1961, initially majoring in physics. He briefly considered philosophy, then switched to mathematics before settling on the emerging field of computer science. His thesis advisor, Michael Harrison, urged his



students to write down all that they learned. Gray took this advice to heart, developing the habit of writing up—and distributing—reports of meetings, trips, and conferences. He soon established two rules for authoring technical papers: “He who types the paper is first author” and “It’s easier to add a co-author than deal with someone’s hurt feelings.” At times, colleagues were unaware of their participation. “I co-authored [one paper] while I wasn’t looking,” quipped Microsoft’s Pat Hella, who joined Tandem in the early 1980s to work with Gray.

“Jim was not dangerous to his colleagues, as some scientists can be,” recalled Mike Blasgen, who was one of Gray’s managers at IBM in the 1970s. “He would not take credit for your ideas. Also, he always had an interesting or provocative insight, so people wanted to be his friend.”

In time, Gray developed a large professional network with which he shared information, both inside and outside of the company he was working for. At Tandem, for example, “he was a great pollinator,” productively sharing ideas between departments, said Wendy Bartlett, now with Hewlett-Packard.

When visiting companies and universities or attending conferences, Gray often sought out students, interns, and young professionals, listening intently

to their research and subtly offering suggestions. “He would not say ‘This is what you must do,’ ” said Alex Szalay of Johns Hopkins University, who had worked with Gray on the Sloan Digital Sky Survey. “He would gently light the way, so that people would find the path themselves.”

Although Gray’s closest colleagues knew firsthand the many hours he typically spent working, many attendees at the tribute were incredulous that any one person could accomplish all that Gray had.

His wife, Donna Lee Carnes, offered some clues. “I don’t think I ever saw Jim procrastinate,” she said. “Writing also came very easily to him. You can get a lot done when you’re focused and fast. He could also read and absorb knowledge very quickly.”

Carnes said her husband had an astounding amount of energy despite the fact that he often slept only five hours a night during the week. “If he was looking for a bug in his program or a product was nearly ready to ship, nothing came between Jim and his work,” Carnes added. “He wouldn’t even stop to eat. Just coffee, sometimes three pots.”

However, after the bug was found and fixed or the product shipped, Gray would enjoy good food, and often sailing, with his wife and friends. When John Nauman, who hired Gray at Tandem, asked the tribute audience how many had gone sailing with Gray, about 100 hands shot into the air.

In announcing his departure from IBM Research in 1980, Gray wrote that he aspired to be “a scholar of computer science,” noting that all fields of scholarship emphasized research, teaching, and service. The Jim Gray Tribute demonstrated that he had clearly achieved that—and much, much more. **□**

Michael Ross writes about science and technology from San Jose, CA.

V viewpoints



DOI:10.1145/1378704.1378711

Peter J. Denning

The Profession of IT Voices of Computing

The choir of engineers, mathematicians, and scientists who make up the bulk of our field better represents computing than the solo voice of the programmer.

ALTHOUGH ENROLLMENTS IN computing degree programs appear to have bottomed out at approximately half of their 2001 level, there is no reassuring upward trend. The industry need for computing professionals will continue to exceed the pipeline by at least one-third for some time to come. Why do low enrollment rates persist in such a good market?

Several key factors influencing low enrollment rates are connected to the myth “CS=programming”—tales of dwindling employment opportunities, negative images of computing work, and inflexible curricula.^{2,3} Reversing this myth can result in considerable progress.

Thirty-five years ago, Edsger Dijkstra reacted to his generation’s version of this myth by declaring himself proud to be a programmer.⁵ Many followed his lead. ACM has been proud: half the A.M. Turing Award winners are in programming, algorithms, and complexity.³ But our internal self-confidence did not dispel the external myth.

Twenty years ago, the ACM and IEEE warned that the myth could become damaging.⁴ Today, the word

“programming” itself generates misunderstandings. Internally, it is broad: design, development, testing, debugging, documentation, maintenance of software, analysis, and complexity of algorithms. Externally, it is narrow: the U.S. Bureau of Labor Statistics defines “programmer” to mean “coder.” Often without realizing it, insiders and outsiders interpret the same words with entirely different meanings. When insiders broadened to object-oriented programming, outsiders thought we narrowed to the Java language.

Ten years ago, we tried another tack. We broadened our view of computing to include information technology (IT),¹ and we defined what it means to be fluent in IT.^{7,8} These works were embraced in high schools and helped generate enrollments in IT but not CS. They have not dispelled the myth.

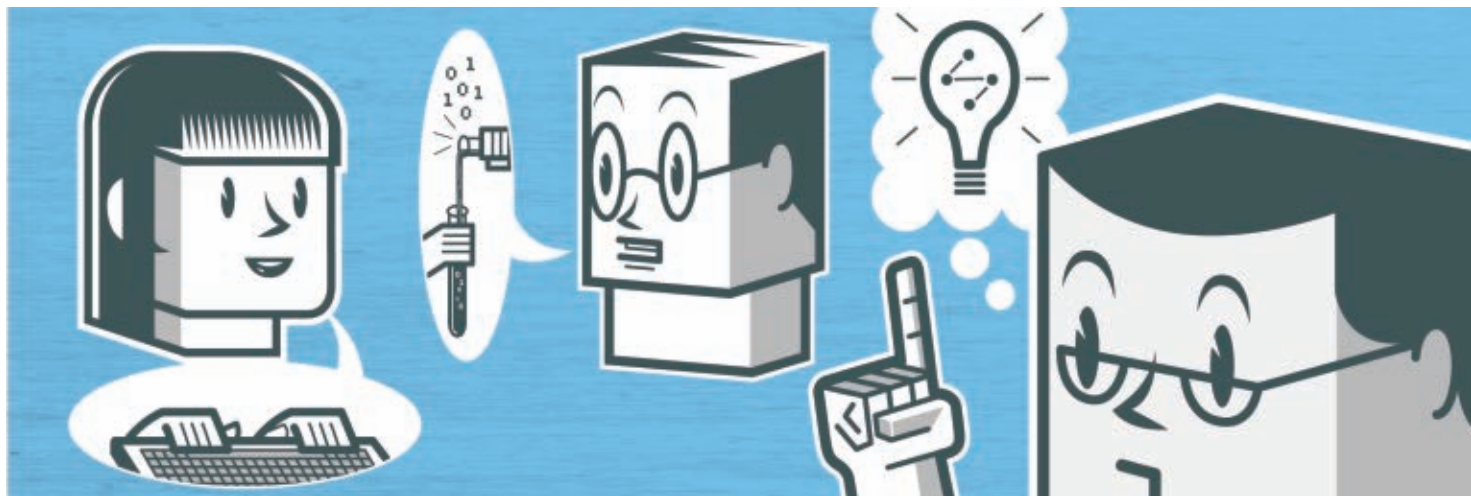
Today, Clay Shirky notes a trend that may help explain the durability of the myth.⁶ The general public is now confronted with an amazing array of powerful tools for the common computational tasks. Many believe they can accomplish what they need as amateurs. Only a few professionals are needed to program all these tools for the many. There is no spe-

cial attraction to being a professional.

How can we communicate the richness of our field and dispel the myth? What if we learn to speak in the voices of the many kinds of computing professionals? The programmer is a solo voice. The whole, loud choir could dispel the perception that the bulk of computing is about programming. The choir might also help make professionalism more attractive by showing our many critical specialties that cannot be done by amateurs.

To speak in a professional’s voice, I immersed myself in the professional’s practice. I spoke of war stories, experiences, ambitions, fears, and everyday things. I sang the joys (and sorrows) of being a professional.

Education philosophers such as John Dewey maintained there are two ways of learning, which can be called “learning-about” and “learning-to-be.” Learning-about means to acquire a description; learning-to-be means to acquire the practice. Learning about carpentry, music, or programming is not the same as being a carpenter, a musician, or a programmer. Programming seems to blur this distinction because programmers build descriptions of al-



gorithms. Programming predisposes us to the “about” side of computing. We are more used to speaking about the principles and ideas of our field than about how individuals experience them. Descriptions of computational methods can be dull and lifeless compared to war stories from professionals who design and use them.

What follows are six voices of computing professionals. I added a seventh, non-professional voice, which I call the Last Voice. It is last not only because it appears at the end of this column, but because it may be the last voice consulted by young people before deciding against computing as a major.

All these voices are already within you. Except the last, just let them speak.

The Programmer

I love programming. I know a lot of languages and can make computers really hum. I do my best work when no one bosses me around—that’s when I am at my most creative. You know, programming is the most fundamental part of computer science. No computer can run without a program. I enable everything else in computing. I have written some history-changing programs. Just think about the software in the Apollo missions—I helped get us to the moon. Think of all those multiplayer virtual reality games—I give a lot of people immense pleasure learning important skills and shooting each other up. I get you safely across the country by helping the air traffic controllers. I get you your food by helping to route the trains and trucks. I gave you your word processor, spreadsheet, PowerPoint applications, and even a few friendly hearted Easter

Eggs. I attacked the Internet with a worm in 1988 and then helped stop the worm and catch the perp. I do a lot of things for you. I know that sometimes you look down at programmers and sometimes you think of us as the computer science equivalent of hamburger-flippers. But we deserve your respect and admiration.

The User

I love using computers. I’m not a computer scientist, and I don’t want to be. I just love using the stuff computer scientists make. Awesome! I get some really spiffy things done with your tools even though I am an amateur. Most of the time, your stuff does not bankrupt me, waste my time, or kill me. My cell phone, instant messages, Web, Internet, Google Earth, Microsoft Office, iTunes, iPod, and the ACM Digital Library. It just goes on and on. I am so grateful to have all this computer stuff. My wants and needs determine what computer scientists can sell, so they often listen to me very carefully. Without

We are more used to speaking about the principles and ideas of our field than about how individuals experience them.

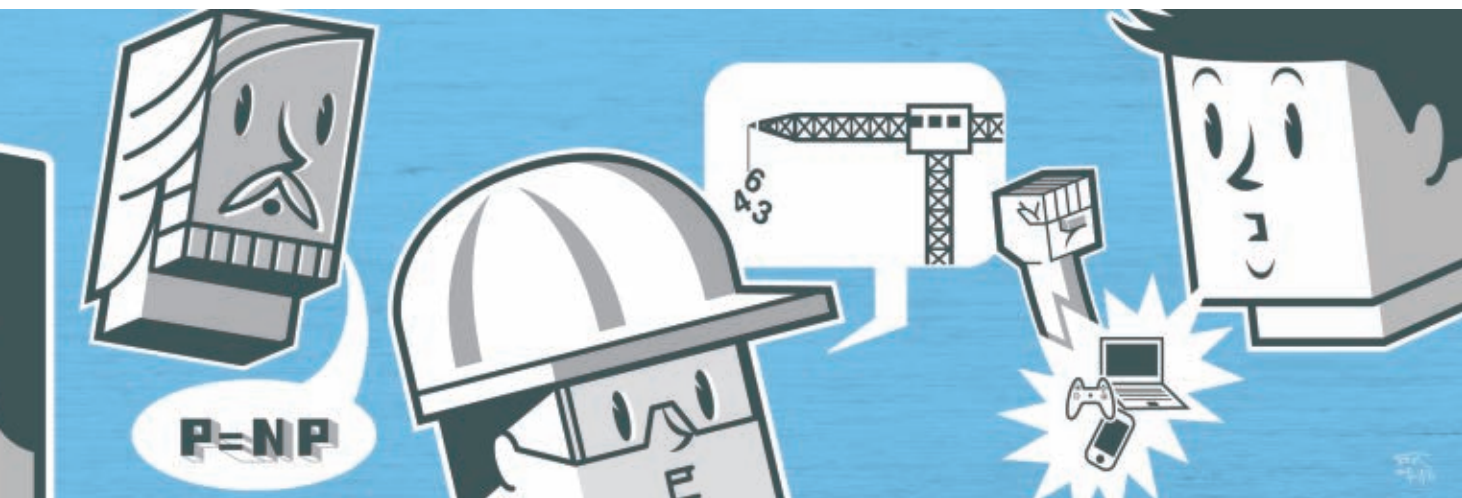
those wants and needs, in fact, I’d be a nobody.

The Computational Thinker

I love problem solving. Not just any old problem solving, but problem solving using algorithms. I love finding ways to apply algorithms I know to solve problems that folks didn’t realize could be solved. It’s such a powerful way to solve problems. All you have to do is think algorithms and—poof!—solutions appear. Sometimes I implement those solutions myself, and sometimes I let my friends the programmers do that. I’ve helped biologists search DNA databases, meteorologists forecast weather, petrologists find oil, oceanographers track ocean currents, linguists teach languages, and tax collectors insert spreadsheet algorithms into the law. Every so often somebody asks if I am a computational scientist. I answer no—while I think about how algorithms can help scientists, I don’t do their science for them. I’m all about thought. One of my greatest successes is to get politicians to think that through their laws they are programmers of national social systems. I’ve got economists thinking they can program the economy with the right policies. Perhaps my greatest triumph is to get people everywhere to think their brains are computers and that everything they do and say is simply an output.

The Mathematician

I love mathematics. I know mathematics sounds pretty abstract to a lot of people. It’s not for everyone. We’ve long been recognized as the language of physics. Now we’ve got the addition-



al recognition of being the language for computation. We help people find representations for real-world things and then prove things about their representations. In computer science we have invented new mathematics for analyzing algorithms. We explained why a binary search is so much faster than a linear search, and why a bubble sort is so much slower than a merge sort. We figured out how to parse programming languages efficiently, making the jobs of programmers so much easier. Our tools help programmers prove that their complicated programs actually work, meaning that users can sleep at night knowing that their airplanes won't crash and their business software won't ruin them. Our biggest triumph has been to show that over 3,000 common problems in science, engineering, and business are so difficult to solve that even the fastest supercomputers would take centuries on simple versions. We call this the $P=NP$ issue. Whoever proves that $P=NP$ would win all the math prizes and the ACM Turing Award. And no, proving $P=NP$ does not boil down to proving $N=1$.

The Engineer

I love building things. My math friends like picturing things in their minds; I like holding things in my hands and putting them through their paces. You tell me what you want, what budget I have, and how much time I have, and I'll find a way to build a computing system that does it. I don't need everything to be figured out mathematically before I can start. I built your operating systems, your networks, your TCP and IP, your air traffic control system,

your banking systems, your game engines, and your search engines. I built your memory chips, your CPUs, your stacks, your graphics displays, your warehouse computers, your BlackBerries, and your iPods. I know how to make software and hardware artifacts reliable, dependable, usable, safe, and secure. I love the smells of solder, motherboards, routers, power supplies, and musty cable racks. Sometimes I even think I can smell rotting bugs in software. I'm so good at doing things faster, cheaper, and better that I keep on giving you Moore's Law year after year.

The Scientist

I love discovering new things about nature. Recently my friends in biology have discovered that DNA transcription is a natural information process. What an amazing discovery. Computation is not an artifact of a computer, it's part of life! My friends in physics, economics, materials, chemistry, meteorology, oceanography, and cosmology are all making similar discoveries. What a great time for collaborations on new discoveries about those natural processes. But that's not all I do. I discovered scientific principles for computing. My scientific analysis guided the design of the first electronic computers. My principle of locality helped us achieve high performance through caching in everything from chips to the Internet. I discovered fast algorithms for throughput and response time of large systems and networks, launching the performance evaluation industry. I brought the experimental method to architecture, program per-

formance improvement, large system design, mathematical software, modeling, and simulation. My greatest triumph in the CS realm has been with artificial intelligence. Now that they have accepted my methods, they are making remarkable advances with machines that mimic human intelligent behavior.

The Last Voice: The Catalog

Students begin by learning the use of computer programming as a problem-solving tool. Topics in procedural programming include expressions, control structures, simple data types, input-output, graphical interfaces, testing, debugging, and programming environments. The student then advances to problem solving with object-oriented programming. Topics include classes, inheritance, packages, collections, exceptions, polymorphism, and recursive thinking. A good deal of time will be spent on programming projects. ■

References

1. Denning, P. Who are we? *Commun. ACM* 44, 2 (Feb. 2001), 15–19.
2. Denning, P. The field of programmers myth. *Commun. ACM* 47, 7 (July 2004), 15–20.
3. Denning, P. Recentering computer science. *Commun. ACM* 48, 11 (Nov. 2005), 15–19.
4. Denning, P. et al. Computing as a discipline. *Commun. ACM* 32, 1 (Jan. 1989), 9–23.
5. Dijkstra, E. The humble programmer. *Commun. ACM* 15, 10 (Oct. 1972), 859–866.
6. Shirky, C. *Here Comes Everybody*. Penguin, 2008.
7. Snyder, L. *Fluency with Information Technology*. Addison-Wesley, 2002.
8. Wing, J. Computational thinking. *Commun. ACM* 49, 3 (Mar. 2006), 33–35.

Peter J. Denning (pjd@nps.edu) is the director of the Cebrowski Institute for Information Innovation and Superiority at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.

© 2008 ACM 0001-0782/08/0800 \$5.00



From the Front Lines

Software Development Amidst the Whiz of Silver Bullets

Software development organizations must accept the inevitability of silver-bullet solution proposals and devise strategies to defend against them.

THE SOFTWARE ENGINEERING landscape remains pockmarked with individuals who continue to disregard Fred Brooks' sage admonitions^a asserting that silver bullets should not be relied upon to solve all woes. The desperate, the pressured, and the ignorant are among those who defiantly worship the silver-bullet gods, pleading for a continuum of the silver-fueled delusions keeping many of their projects alive. It is difficult to be overly critical of those who have succumbed to silver bullets, however, because the software engineering space is being strafed with them as never before. In fact, even the most savvy must occasionally liken themselves to the infamous Neo in the film *The Matrix* and gyrate wildly to avoid being stricken by the many silver bullets whizzing by.

Veterans of the software industry will attest to having seen a number of silver bullets come and go during their careers. The argentum^b projectiles of yesteryear, such as OO, high-level software languages, and integrated development environments, are now obvious to have been only low-grade alloys compared to the fine silver being discharged today. Some of today's silver bullets have demonstrated an unparalleled ability to provide implicit value to

a Brooks, F.P., Jr. No silver bullet, essence and accidents of software engineering. *Computer Magazine* (Apr. 1987).

b The Latin word for silver and the basis of the periodic symbol: Ag.



artifacts just because they were created using a particular technology while others have demonstrated the power to shift the responsibilities associated with long-established engineering disciplines to other organizations. Only the passage of time will reveal the new and amazing capabilities of future silver bullets that have yet to whiz by.

Getting back to today's silver bullets, though, I was recently reviewing a software design package that cor-

rectly paid much-needed attention to the objective of supporting configurable runtime behavior. As opposed to simply documenting how the design would accommodate this desired configurability, however, the design description also included a compelling assertion a number of times: "The configuration data will be stored in XML." What on earth did this have to do with anything? Should I have been relieved that some form of irregular

Egyptian hieroglyph had not instead been chosen as the representation choice for this configuration data?

Still reeling from the potentially powerful implications of what I had read, I began to wonder if any textual content expressed in XML would somehow lead people to believe it to be of high pedigree, divine origin, and having some implicit warranty of accuracy or correctness. I decided to test this premise and composed an email message to my 12-year-old daughter hoping to sway her on an opinion she had been stubbornly adhering to in the past—see Figure 1. I was hopeful that if she were to read my message within the context of XML, our long-standing dispute would finally get some much-needed resolution. Unfortunately, things did not work out as I had hoped, and my ploy only served to further reinforce her unyielding position on my standing among other fathers.

XML is not the only silver bullet in the software engineering toolkit to which value seems to be implicit by mere usage. The fact that a diagram has been created using UML leads some to believe that the associated design is guaranteed to be implementable and ready for development even if the laws of physics were ignored as constraints. Had my daughter not already convincingly dashed the notion of implicit technological sanctity, I certainly would be tempted to create a yoo-mel^c sequence diagram for my wife detailing the steps I plan to take someday to be more sensitive, a better listener, and less of a slob. If she were to see my plan captured in UML, perhaps she would be more inclined to believe the sincerity of my intentions?

No discourse on silver bulletry would be complete without giving due recognition to a current favorite: Web services. “That’s just a Web service...” is a phrase often spoken so convincingly that it is hard to believe that there are not Web services already available with which to accommodate all known functional needs. After all, there are already Web services available that provide the stock price for any ticker symbol or the ZIP code for any city, how difficult could it be to

create a new one that simply returns numbers that violate the Goldbach Conjecture?^d Considerations for strict temporal determinism, sporadic network availability, or the fact that a Web service’s signature does not support one’s workflow needs are unimportant amidst the whiz of silver bullets.

Although vulnerability to error and productivity impacts of working directly with XML inspired the innovation of technologies such as WSDL with which to improve the usability of Web services, some developers feel that their usage short-circuits the unparalleled maintainability and flexibility properties offered by method signatures as shown in Figure 2.

Why should anyone subject themselves to the brittleness of specialized methods when a single method can accommodate currently required behaviors and any that may only be known in the future? Additional benefits of offering methods with `DoAnything()` signatures include freeing designers from the burdens of time-consuming negotiation with prospective service users and not having to be bothered with recompilation issues that typically accompany usage of more strongly typed interfaces. It would serve as a great justice if the providers of methods such as `DoAnything()` also assumed their associated liabilities. The unfortunate reality, however, is that the method’s users typically have to

endure the associated liabilities in the form of increased testing and integration costs as the result of misuse being virtually undetectable at compile time. In response to suggestions that a `DoAnything()` method should be redesigned to take advantage of strong typing, it is not uncommon to hear its designers assert something of the sort, “An `XMLCommand` is a strong type, let’s see you try to use an integer argument in its place!”.

The challenges of software development are difficult enough without also having to endure the ricochet of silver bullets strafing the organizations responsible for engineering the products that software development itself relies upon. For example, some systems engineers have discovered that usage of UML greatly simplifies efforts that their predecessors unnecessarily struggled with in pre-UML days. As opposed to having to devote significant efforts to the consideration of constraints such as network bandwidth, processor speeds, and the speed of light when developing system architectures, such annoyances are now overcome by creating stacks of UML diagrams that abstract these details away as uninteresting implementation issues. I wonder if there is any way for us software guys to further kick this problem down the road by convincing testers that their jobs would be much easier if they validated UML models instead of software?

One of my favorite television commercials of all time helps characterize the situation. While sitting in a police station, a crime victim is providing a

d A long-unsolved math problem asserting that all even positive integers ≥ 4 can be expressed as the sum of two primes.

Figure 1: Sample message text expressed in XML.

```
<message _from _Dad>
  <addressee> Alanah </addressee>
  <message> Hi Sweetie, I am not the weirdest Dad of all the kids
    in your whole school.
    Love, Dad.
  </message>
</message _from _Dad>
```

Figure 2: Sample method signature.

```
XMLResult DoAnything (XMLCommand Arguments)
{
  .....
}
```

c Spoken in a drawl, a euphemism for insane usage of the UML.

detailed description of his alleged perpetrator to a police artist who appears to be diligently sketching a corresponding likeness on an unseen tablet of paper. After some time, the police artist believes that the likeness he has captured is sufficiently ready for the victim to assess its accuracy. Amidst a rising crescendo of expectation, the artist reveals his drawing to the victim. Shockingly, the drawing contains only a human stick figure that would be considered primitive by even kindergarten standards. All of the critical detail is missing from the sketch upon which to base future work, just like when *CreateWorldPeace* or *IncreaseTheSpeedOfLight* use cases are delivered to software engineering organizations for implementation. Abstraction has a whole new meaning amidst the whiz of silver bullets!

Silver bullets of the past and present share a number of common properties that will also likely apply to the bullets only now forming in the mental foundries of the fantasy minded. They defy the laws of physics, they are not bound-

ed by cost, they are not constrained by time, and they seem to rob otherwise intelligent people of their common sense. Their usage is typically accompanied by postponing engineering efforts to a time later in the product life cycle, shirking responsibilities to other organizations, and blatant disregard for reality. And probably the most consistent property associated with silver bullets is that the people who promote or endorse them have generally never been software developers nor have they directly contributed to the delivery of a successful program. With Fred Brooks' well-known admonitions aside, it is startling that the failure of past silver bullets is insufficient to make people very wary of them today.

Barring future events similar to the Hunt brothers' failed attempt to corner the silver market in 1980, silver futures appear to be bullish. The supply of fine-grade silver required to manufacture the next generation of silver bullets is projected to meet future demand so the sound of their whiz will not subside anytime soon. As a result,

software organizations must accept that silver bullets will be a part of their future and should prepare strategies to defend against them rather than to assume their demise. The only plausible defense strategy against silver bullets that I have been able to think of is to assemble an engineering staff that has a natural affinity for eluding these projectiles and one with an innate terror of them. Assembling such a staff, however, would involve an undertaking that is currently quite unpopular: outsourcing. As opposed to China or India, however, my outsourcing plan would focus on a small town in Romania. For it is only in Transylvania where one can assemble a team of those certain someones having an innate fear of silver bullets...werewolves. □

Alex E. Bell (alex.e.bell@boeing.com) is a software architect with The Boeing Company and author of "Death by UML Fever."

A previous version of this material appeared in the June 2006 issue of *ACM Queue*.

© 2008 ACM 0001-0782/08/0800 \$5.00

ACM Journal on Emerging Technologies in Computing Systems



◆ ◆ ◆ ◆ ◆

This quarterly publication provides comprehensive coverage of innovative work in the specification, design analysis, simulation, verification, testing, and evaluation of computing systems constructed out of emerging technologies and advanced semiconductors. Topics include, but are not limited to: Logic Primitive Design and Synthesis; System-Level Specification, Design and Synthesis; Software-Level Specification, Design and Synthesis; and Mixed-Technology Systems.

◆ ◆ ◆ ◆ ◆

<http://jetc.acm.org/>



Education

Paving the Way for Computational Thinking

Drawing on methods from diverse disciplines—including computer science, education, sociology, and psychology—to improve computing education.

TEACHING EVERYONE ON campus to program is a noble goal, put forth by Alan Perlis in 1962. Perlis, who was awarded the first ACM A.M. Turing Award, said that everyone should learn to program as part of a liberal education. He argued that programming was an exploration of process, a topic that concerned everyone, and that the automated execution of process by machine was going to change everything. He saw programming as a step toward understanding a “theory of computation,” which would lead to students recasting their understanding of a wide variety of topics (such as calculus and economics) in terms of computation.⁴

Today, we know that Perlis was prescient—the automated execution of process *is* changing how professionals of all disciplines think about their work. As Jeanette Wing has pointed out, the metaphors and structures of computing are influencing all areas of science and engineering.⁶ Computing professionals and educators have the responsibility to make computation available to thinkers of all disciplines.

Part of that responsibility will be met through formal education. While a professional in another field may be able to use an application with little training, the metaphors and ways of thinking about computing must be explicitly taught. To teach computational thinking to everyone on campus

may require different approaches than those we use when we can assume our students want to become computing professionals. Developing approaches that will work for all students will require us to answer difficult questions like what do non-computing students understand about computing, what

will they find challenging, what kinds of tools can make computational thinking most easily accessible to them, and how should we organize and structure our classes to make computing accessible to the broad range of students.

Through a few brief examples, I will show in this column how these



ACM Digital Library

www.acm.org/dl



The Ultimate Online INFORMATION TECHNOLOGY Resource!

- NEW! Author Profile Pages
- Improved Search Capabilities
- Over 40 ACM publications, plus conference proceedings
- 50+ years of archives
- Advanced searching capabilities
- Over 2 million pages of downloadable text

Plus over one million bibliographic citations are available in the ACM Guide to Computing Literature

To join ACM and/or subscribe to the Digital Library, contact ACM:

Phone: 1.800.342.6626 (U.S. & Canada)
+1.212.626.0500 (Global)

Fax: +1.212.944.1318

Hours: 8:30 a.m.-4:30 p.m., EST

Email: acmhelp@acm.org

Join URL: www.acm.org/joinacm

Mail: ACM Member Services

General Post Office

PO Box 30777

New York, NY 10087-0777 USA

questions are being addressed by researchers in the field of computing education research. Researchers in computing education draw on both computer science and education—neither field alone is sufficient. While we computer scientists understand computing from a practical, rational, and theoretical perspective, questions about education are inherently human questions. Humans are often impractical, irrational, and difficult to make predictions or proofs about. Computing education researchers are using experimentation and design to demonstrate we can address important questions about how humans come to understand computing, and how we can make it better. Research in computing education will pave the way to make “computational thinking” a 21st century literacy that we can share across the campus.

Understanding Computing Before Programming

A research theme in the early 1980s was how to design programming languages so they would be more like natural languages. An obvious question, then, is how people specify processes in natural language. Lance A. Miller asked his study participants to specify file manipulation tasks for another person. A task might be “Make a list of employees who have a job title of photographer and who are rated superior, given these paper files.” Miller studied the language used in his participants’ descriptions.²

One of Miller’s surprises was how rarely his participants explicitly specified any kind of control flow. There was almost no explicit looping in any of their task descriptions. While some tested conditions (“IF”), none ever specified an “ELSE.” He found this so

Figure 1: Traditional conditional structure.

```
if (value < 10)
then value = value + 10;
else sum = sum + value;
end if
```

Figure 2: New conditional structure.

```
if (value < 10): value = value + 10;
not (value < 10): sum = sum + value;
end (value < 10)
```

surprising that he gave a second set of participants an example task description, without looping and no ELSE specification. The second set of participants easily executed the task description. When asked what they were doing if the condition was not met, or if data was exhausted, they replied (almost unanimously, Miller reports), “Of course, you just check the next person, or if there are no more, you just go on.”

Miller’s results predict some of the challenges in learning to program—challenges that are well-known to teachers of introductory classes today. While process descriptions by novices tend not to specify what to do under every condition, computers require that specificity. Miller’s results suggest what kinds of programming languages might be easier for novices. Programming languages like APL and MATLAB, and programming tools for children like Squeak’s eToys use implicit looping, as did the participants in Miller’s studies.

Twenty years later, John Pane and his colleagues at Carnegie Mellon University revisited Miller’s questions, in new contexts.³ In one experiment, Pane showed his subjects situations and processes that occur in a Pacman game, then asked how they would specify them. The subjects responded with explanations like, “When Pacman gets all the dots, he goes to the next level.” Like Miller, Pane found that participants rarely used explicit looping and always used one-sided conditionals. Pane went further, to characterize the style of programming that the participants used. He found that over half of the participants’ task statements were in the form of production rules, as in the example. He also saw the use of constraints and imperative statements, but little evidence of object-oriented thinking. Participants did talk about accessing behaviors built into an entity, but rarely from the perspective of that entity; instead, it was from the perspective of the player or the programmer. He found no evidence of participants describing categories of entities (defining classes), inheritance, or polymorphism.

Pane’s results suggest that object-oriented thinking is not “natural,” in the sense of being characteristic of novices’ task descriptions. Since ob-

jects are the foundation of most modern software today, his results point out where we can expect to find challenges in explaining objects to students. Both Miller's and Pane's results encourage us to think how we might design languages for novices that play to their natural ways of thinking about specifying computation, like the use of event-based programming in MIT's Scratch.

In the last four years, a multinational group of researchers has explored "Commonsense Computing": what do our students know before we teach them? Given a complex task, how do people without programming knowledge specify an algorithm for that task? In one paper, Lewandowski et al.¹ explore concurrency—in a complex task of multiple box offices selling tickets for a theater, how well do non-programming students avoid selling the same seat twice? The results showed that 97 solutions (69% of the total, drawn from five institutions) were correct; only 31% of the solutions (45% of the correct solutions) were distributed, so teachers of algorithms classes need not worry about being put out of business. Non-computing students do not naturally come up with the elegant solutions that computer scientists have devised. However, these results suggest that students can "naturally" think about concurrency correctly. Problems with implementing concurrent programs might stem more from the challenges in specifying those algorithms in current programming languages, rather than from the complexity of the algorithms themselves.

Redesigning Programming Languages

Both Pane's and Miller's results make suggestions about the design of programming languages if the goal is to make computational ideas more accessible to novices. Testing new forms of programming languages was an area of active exploration by Thomas R.G. Green, Elliot Soloway, and others.

In one paper, Green and his colleagues explored alternatives to the traditional conditional structure.⁵ A typical structure might look like the structure shown in Figure 1. They tested a new structure where this would be written as shown in Figure 2. This new structure makes explicit the condition

Research in computing education will pave the way to make "computational thinking" a 21st century literacy that we can share across the campus.

for the execution of each clause of the condition. Green and his colleagues found that novices were able to correct mistakes using the second form 10 times faster than programs using the first form.

Miller and Pane found that their participants simply never used an else clause. Instead, it seemed obvious ("of course") what to do when the tested condition wasn't true. Miller's and Pane's subjects were doing something different than Green's. Writing a task description is different than reading and fixing a task description. Green's results complement Miller's and Pane's. Novices do not naturally write the else clause—they think it's obvious what to do if the test fails. However, conditionals in programs are not always obvious, and it's easier for the novices trying to read those programs if the conditions for each clause's execution are explicit.

Paving the Way for "Computational Thinking" For All

To make "computational thinking" accessible to students across the entire campus, we need to understand how to teach computing better. Computing education researchers explore how humans come to understand computing, and how to improve that understanding. Computing education research is a close cousin to human-computer interaction, since HCI researchers explore how humans interact with computing and how to improve that interaction. Computing education researchers

have found a home in the International Computing Education Research (ICER) workshop (whose fourth annual meeting will be held this September in Sydney, Australia; see www.newcastle.edu.au/conference/icer2008/) and in journals like *Computer Science Education* and *Journal on Educational Resources in Computing*.

Computing education research draws on a variety of disciplines to make computing education better. Social scientists like Jane Margolis, Lecia Barker, and Carsten Schulte help us to understand how students experience our classes (which often differs from what we might expect as teachers) and how we can change our classes to make them more successful for all students. Computing education researchers draw on methods from education, sociology, and psychology in order to measure learning about computing and understand the factors that influence that learning. By making computing education better, we can broaden access to computing ideas and capabilities. When we can teach every student programming and the theory of computation in a way that makes sense to them for their discipline, we will see how ubiquitous understanding of computing will advance the entire academy, just as Perlis predicted over 45 years ago. ■

References

- Lewandowski, G. et al. Commonsense computing (episode 3): Concurrency and concert tickets. In *Proceedings of the Third International Workshop on Computing Education Research* (2007), 133–144.
- Miller, L.A. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal* 29, 2 (1981), 184–215.
- Pane, J.F., Ratanamahatana, C., and Myers, B.A. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies* 54 (2001), 237–264.
- Perlis, A. The computer in the university. In M. Greenberger, Ed., *Computers and the World of the Future*, MIT Press, Cambridge, MA, 1962, 180–219.
- Sime, M.E., Arblaster, A.T., and Green, T.R.G. Structuring the programmer's task. *Journal of Occupational Psychology* 50 (1977), 205–216.
- Wing, J. Computational thinking. *Commun. ACM* 49, 3 (Mar. 2006), 33–35.

Mark Guzdial (guzdial@cc.gatech.edu) is a professor in the College of Computing at Georgia Institute of Technology in Atlanta, GA.

The *Communications* "Education" column will feature commentary on education issues, presenting research results and opinions that inform how the challenges of computing education can be best addressed.

Viewpoint

Envisioning the Future of Computing Research

Advances in computing have changed our lives—the Computing Community Consortium aims to help the research community continue that lineage.

HOW CAN WE work together to establish, articulate, and pursue compelling visions for our field—visions that will shape the intellectual future of the field, that will catalyze research investment and public support, and that will attract the best and brightest minds of a new generation?

The National Science Foundation asked the Computing Research Association to create the Computing Community Consortium (CCC) to address this challenge. The mechanics of the CCC have been described elsewhere;⁵ in this column, I focus on the substance.

Computing has Made Extraordinary Progress

William Shockley, Walter Brattain, and John Bardeen invented the transistor at Bell Laboratories in 1947, just over 60 years ago. Jack Kilby at Texas Instruments and Bob Noyce at Fairchild Semiconductor demonstrated the integrated circuit only 50 years ago, in 1958. It was 1965—just a bit more than 40 years ago—when Gordon Moore described what is now universally referred to as “Moore’s Law.”

Today, the computational power of an early mainframe can be found in an electronic greeting card, and the computational power that guided Apollo 11 to the moon is contained in a Furby electronic toy. There are more than one billion PCs, and nearly that

many Internet hosts.

It was only 10 years ago that Deep Blue—a supercomputer by any definition—defeated world chess champion Garry Kasparov. Today, thanks more to progress in software than to progress in hardware, you can download for your PC a chess engine with a rating 10% higher than any human player. Most of the “futurist scenarios” described when *Time* magazine featured the computer as “Machine of the Year” 25 years ago have been realized, including computer-controlled tailoring using laser-scanning, robots performing domestic chores, embedded systems that people don’t realize are computers at all.

Advances in computing are changing the way we live, work, learn, and communicate. These advances are driving advances in nearly all other fields and are significantly influencing the U.S. economy—not just through the growth of the IT industry, but even more importantly, through productivity growth across all sectors.

Research has Laid the Foundation

Almost every aspect of computing that is integral to our lives today can trace its roots, at least in substantial part, to federally sponsored research. In 1995, the National Academies’ “Brooks-Sutherland Report”² traced the lineage of a number of billion-dollar sub-sectors of the computing indus-

try: timesharing, computer graphics, networking (LANs and the Internet), personal workstation computing, windows and the graphical user interface, RISC architectures, modern integrated circuit design, RAID storage, and parallel computing. In each case, the role of federally sponsored research was clear.

The panel conducting this study (I was one of the 12 members) lamented our inability to identify new ideas that might someday be comparably influential. But eight years later, the National Academies did a reprise of the study⁴ and noted entertainment technology, data mining, portable communication, the Web, speech recognition, and broadband last mile as new billion-dollar subsectors whose roots could be traced, at least in substantial part, to federally sponsored research. (The figure on the next page shows the approximate time frame from concept to billion-dollar industry.)

While we may not be sure which they are, there surely are technologies in our laboratories today that will have comparable impact a decade from now.

The Future is Full of Opportunity

Several months ago, the National Academy of Engineering unveiled 14 “Grand Challenges for Engineering” for the 21st century.³ The majority of these “Grand Challenges” for *all* of engineer-

ing have either substantial or preponderant computer science content:

- ▶ Secure cyberspace
- ▶ Enhance virtual reality
- ▶ Advance health information systems
- ▶ Advance personalized learning
- ▶ Engineer better medicines
- ▶ Engineer the tools of scientific discovery
- ▶ Reverse-engineer the brain
- ▶ Prevent nuclear terror (to a great extent a sensor network and data mining problem)

These are, in every way, visions that can shape the intellectual future of our field, catalyze research investment and public support, and attract the best and brightest minds of a new generation. And there are many more such visions:

- ▶ Create the future of networking
- ▶ Empower the developing world through appropriate information technology
- ▶ Design automobiles that don't crash
- ▶ Build truly scalable computing systems
- ▶ Engineer advanced "robotic prosthetics"—the field of Neurobotics
- ▶ Instrument your body as thoroughly as your automobile
- ▶ Engineer biology (synthetic biology)
- ▶ Achieve quantum computing

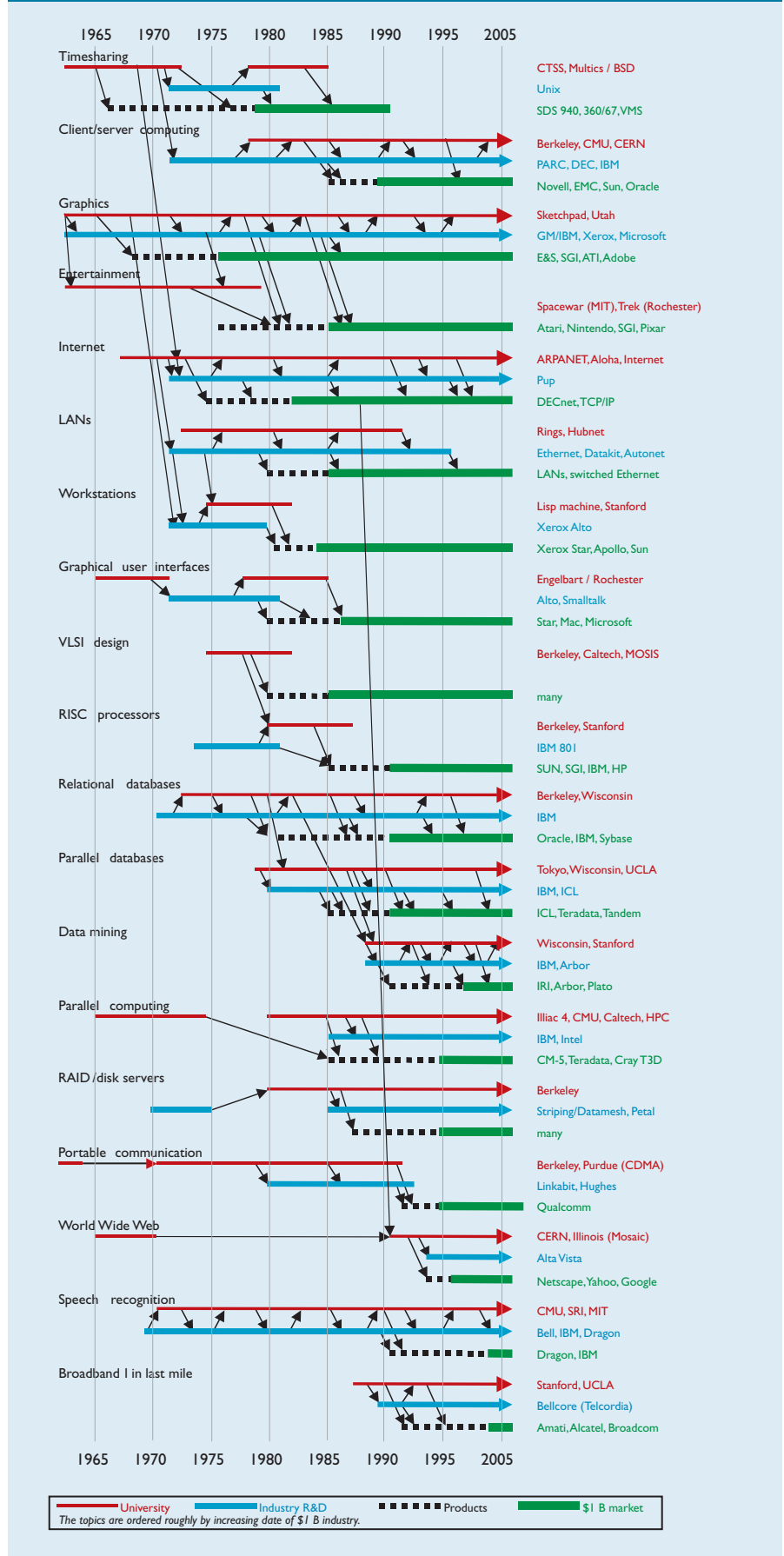
It is very difficult to imagine a field with greater opportunity to change the world.

The Role of the Computing Community Consortium

The role of the Computing Community Consortium is to help our field "put the meat on the bones" of visions such as these. For each of these visions—and for others—we must work together to build a research community, lay out a research roadmap, and acquire momentum.

One way in which CCC is doing this is to sponsor a series of workshops on various topics: thus far, "big data computing," "cyber-physical systems," visions for theoretical computer science, the future of robotics, and network science and engineering. CCC is actively soliciting proposals for additional workshops from members of the research community.

The "tire tracks" diagram illustrates time from concept to billion-dollar industry.



The participants in these workshops are primarily researchers. The workshops also involve representatives of funding agencies—critical to transitioning research visions into funded programs. Often they also involve industrial participants. A recent example of success is CCC’s “Big Data Computing Study Group.” In late March 2008, two workshops were held in Sunnyvale, CA. The first was the “Hadoop Summit,” whose goal was to build a community of users of Hadoop, an open-source version of Google’s MapReduce system¹ for distributing computations across clusters of thousands of nodes. The second was the “Data-Intensive Scalable Computing Symposium,” whose goal was to build a community of researchers concerned with various issues related to “big-data computing” (slides, videos, and summaries are linked from the CCC Web site; www.cra.org/ccc/). Both of these community-building exercises were successful. And, as a result of preliminary work done by the core group of organizers of this effort, Google, IBM, and Ya-

hoo! have made large-scale clusters available to the academic community for education and research, and the National Science Foundation has announced its CluE (Cluster Exploratory) research initiative. There is no magic here—it takes dedicated individuals to make things happen. But CCC can be an enabler.

A number of other CCC activities are described on CCC’s Web site, which includes descriptions of various grand challenge problems and a blog devoted to discussions of research visions for the field. More broadly, CCC is working along with other organizations to “get the word out” regarding our field. I encourage you to become engaged. Participate in the CCC research visions blog (www.cccb.org/). Join with colleagues to propose a workshop to chart a compelling vision for future of your subfield.

Computer science has accomplished so much, and there is so much additional exciting work to do. The opportunities are truly extraordinary. It’s up to us to seize these opportunities. ■

References

1. Dean, J. and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSDI '04)*, (San Francisco, CA, Dec. 2004); labs.google.com/papers/mapreduce.html.
2. *Evolving the High Performance Computing and Communications Initiative to Support the Nation's Information Infrastructure*. Computer Science and Telecommunications Board, National Research Council, 1995; www7.national-academies.org/cstb/pub_hpcci.html.
3. *Grand Challenges for Engineering*. National Academy of Engineering; www.engineeringchallenges.org.
4. *Innovation in Information Technology*. Computer Science and Telecommunications Board, National Research Council, 2003; www7.national-academies.org/cstb/pub_itinnovation.html.
5. The Computing Community Consortium: Who, what, when, where, why, and how. *Computing Research News* 20, 1 (Jan. 2008); www.cra.org/CRN/issues/0801.pdf.

Ed Lazowska (lazowska@cs.washington.edu) is the Bill & Melinda Gates Chair in Computer Science & Engineering at the University of Washington and the chair of the Computing Community Consortium.

© 2008 ACM 0001-0782/08/0800 \$5.00

ACM Transactions on Internet Technology



This quarterly publication encompasses many disciplines in computing—including computer software engineering, middleware, database management, security, knowledge discovery and data mining, networking and distributed systems, communications, and performance and scalability—all under one roof. TOIT brings a sharper focus on the results and roles of the individual disciplines and the relationship among them. Extensive multi-disciplinary coverage is placed on the new application technologies, social issues, and public policies shaping Internet development.

<http://toit.acm.org/>

Interview

Donald Knuth: A Life's Work Interrupted

In this second of a two-part interview by Edward Feigenbaum, we find Knuth, having completed three volumes of The Art of Computer Programming, drawn to creating a system to produce books digitally.

Don switches gears and for a while becomes what Ed Feigenbaum calls "The World's Greatest Programmer."

There was a revolutionary new way to write programs that came along in the 1970s called "structured programming." At Stanford we were teaching students how to write programs, but we had never really written more than textbook code ourselves in this style. Here we are, full professors, telling people how to do it, but having never done it ourselves except in really sterile cases with no real-world constraints. I was itching to do it. Thank you for calling me the world's greatest programmer—I was always calling myself that in my head. I love programming, and so I loved to think that I was doing it as well as anybody. But the fact is the new way of programming was something that I hadn't had time to invest much effort in.

The motivation is his love affair with books...

That goes very deep. My parents disobeyed the conventional wisdom by teaching me to read before I entered kindergarten. I have a kind of strange love affair with books going way back. I also had this thing about the appearance of books. I wanted my books to have an appearance that other readers would treasure, not just appreciate because there were some words in there.

For Part I of this interview, see *Communications*, July 2008, page 35.



...and what had happened to his books.

Printing was done with hot lead in the 1960s, but they switched over to using film in the 1970s. My whole book had been completely re-typeset with a different technology. The new fonts looked terrible! The subscripts were in a different style from the large letters, for example, and the spacing was very bad. You can look at books printed in the early 1970s and almost everything looked atrocious in those days. I couldn't stand to see my books so ugly. I spent all this time working on them, and you can't be proud of something that looks hopeless. I was tearing out my hair.

At the very same time, in February 1977, Pat Winston had just come out

with a new book on artificial intelligence, and the proofs of it were being done at III [Information International, Incorporated] in Southern California. They had a new way of typesetting using lasers. All digital, all dots of ink. Instead of photographic images and lenses, they were using algorithms, bits. I looked at Winston's galley proofs. I knew it was just bits, but they looked gorgeous.

I canceled my plan for a sabbatical in Chile. I wrote saying "I'm sorry; instead of working on Volume 4 during my sabbatical, I'm going to work on typography. I've got to solve this problem of getting typesetting right. It's only zeros and ones. I can get those dots on the page, and I've got to write this program." That's when I became an engineer. I did sincerely believe that it was only going to take me a year to do it.

But, in fact, it was to be a 10-year project. The prototype user was Phyllis Winkler, Don's secretary.

Phyllis had been typing all of my technical papers. I have never seen her equal anywhere, and I've met a lot of really good technical typists. My thought was definitely that this would be something that I would make so that Phyllis would be able to take my handwritten manuscripts and go from there.

The design took place in two all-nighters. I made a draft. I sat up at the AI lab one evening and into the early morning hours, composing what I thought would be the specifications

of a language. I looked at my book and I found excerpts from several dozen pages where I thought it gave all the variety of things I need in the book. Then I sat down and I thought, well, if I were Phyllis, how would I like to key this in? What would be a reasonable format that would appeal to Phyllis, and at the same time something that as a compiler writer I felt I could translate into the book? Because TeX is just another kind of a compiler; instead of going into machine language you're going into words on a page. That's a different output language, but it's analogous to recognizing the constructs that appear in the source file.

The programming turned out to be harder than he thought.

I showed the second version of the design to two of my graduate students, and I said, "Okay, implement this, please, this summer. That's your summer job." I thought I had specified a language. To my amazement, the students, who were outstanding students, did not complete it. They had a system that was able to do only about three lines of TeX. I thought, "My goodness, what's going on? I thought these were good students." Later I changed my attitude, saying, "Boy, they accomplished a miracle." Because going from my specification, which I thought was complete, they really had an impossible task, and they had succeeded wonderfully with it. These guys were actually doing great work, but I was amazed that they couldn't do what I thought was just sort of a routine task. Then I became a programmer in earnest, I had to do it.

This experience led to general observations about programming and specifications.

When you're doing programming, you have to explain something to a computer, which is dumb. When you're writing a document for a human being to understand, the human being will look at it and nod his head and say, "Yeah, this makes sense." But there are all kinds of ambiguities and vagueness that you don't realize until you try to put it into a computer. Then all of a sudden, almost every five minutes as you're writing the code, a question comes up that wasn't addressed in the

specification. "What if this combination occurs?" It just didn't occur to the person writing the design specification. When you're faced with doing the implementation, a person who has been delegated the job of working from a design would have to say, "Well, hmm, I don't know what the designer meant by this."

It's so hard to do the design unless you're faced with the low-level aspects of it, explaining it to a machine instead of to another person. I think it was George Forsythe who said, "People have said you don't understand something until you've taught it in a class. The truth is you don't really understand something until you've taught it to a computer, until you've been able to program it." At this level, programming was absolutely important.

When I got to actually programming TeX, I had to also organize it so that it could handle lots of text. I had to develop a new data structure in order to be able to do the paragraph coming in text and enter it in an efficient way. I had to introduce ideas called "glue," and "penalties," and figure out how that glue should disappear at bound-

"I wake up in the morning with an idea, and it makes my day to think of adding a couple of lines to my program. It gives me a real high. It must be the way poets feel, or musicians, or painters. Programming does that for me."

aries in certain cases and not in others. All these things would never have occurred to me unless I was writing the program.

Edsger Dijkstra gave this wonderful Turing lecture early in the 1970s called "The Humble Programmer." One of the points he made in his talk was that when they asked him in Holland what his job title was, he said, "Programmer," and they said, "No, that's not a job title. You can't do that; programmers are just coders. They're people who are assigned like scribes were in the days when you needed somebody to write a document in the Middle Ages." Dijkstra said no, he was proud to be a programmer. Unfortunately, he changed his attitude completely, and I think he wrote his last computer program in the 1980s.

I checked the other day and found I wrote 35 programs in January, and 28 or 29 programs in February. These are small programs, but I have a compulsion. I love to write programs. I think of a question that I want to answer, or I have part of my book where I want to present something, but I can't just present it by reading about it in a book. As I code it, it all becomes clear in my head. The fact that I have to translate my knowledge of this method into something that the machine is going to understand forces me to make that knowledge crystal-clear in my head. Then I can explain it to somebody else infinitely better. The exposition is always better if I've implemented it, even though it's going to take me more time.

It didn't occur to me at the time that I just had to program in order to be a happy man. I didn't find my other roles distasteful, except for fundraising. I enjoyed every aspect of being a professor except dealing with proposals, which was a necessary evil. But I wake up in the morning with an idea, and it makes my day to think of adding a couple of lines to my program. It gives me a real high. It must be the way poets feel, or musicians, or painters. Programming does that for me.

The TeX project led to METAFONT for the design of fonts. But it also wasn't smooth sailing.

Graphic designers are about the nicest people I've ever met in my life. In

“I found that writing software was much more difficult than anything else I had done in my life. I had to keep so many things in my head at once. I couldn’t just put it down and start something else. It really took over my life during this period.”

the spring of 1977, I could be found mostly in the Stanford Library reading about the history of letter forms. Before I went to China that summer I had drafted the letters for A to Z.

One of the greatest disappointments in my whole life was the day I received in the mail the new edition of *The Art of Computer Programming* Volume 2, which was typeset with my fonts and which was supposed to be the crowning moment of my life, having succeeded with the TeX project. I think it was 1981, and I had the best typesetting equipment, and I had written a program for the 8-bit microprocessor inside. It had 5,000 dots-per-inch, and all the proofs coming out looked good on this machine. I went over to Addison-Wesley, who had typeset it. There was the book, and it was in the familiar beige covers. I opened the book up and I’m thinking, “Oh, this is going to be a nice moment.” I had Volume 2, first edition. I had Volume 2, second edition. They were supposed to look the same. Everything I had known up to that point was that they would look the same. All the measurements seemed to agree. But a lot of distortion goes on, and our optic nerves aren’t linear. All kinds of things were happening. I



burned with disappointment. I really felt a hot flash, I was so upset. It had to look right, and it didn't, at that time. I'm happy to say that I open my books now and I like what I see. Even though they don't match the 1968 book exactly, the way they differ are pleasing to me.

What it was like writing TeX.

Structured programming gave me a different feeling from programming the old way—a feeling of confidence that I didn't have to debug something immediately as I wrote it. Even more important, I didn't have to mock-up the unwritten parts of the program. I didn't have to do fast prototyping or something like that, because when you use structured programming methodology you have more confidence that it's going to be right, that you don't have to try it out first. In fact, I wrote all of the code for TeX over a period of seven months, before I even typed it into a computer. It wasn't until March 1978 that I spent three weeks debugging everything I had written up to that time.

I found that writing software was much more difficult than anything else I had done in my life. I had to keep so many things in my head at once. I couldn't just put it down and start something else. It really took over my life during this period. I used to think there were different kinds of tasks: writing a paper, writing a book, teaching a class, things like that. I could juggle all of those simultaneously. But software was an order of magnitude harder. I couldn't do that and still teach a good Stanford class. The other parts of my life were largely on hold, including *The Art of Computer Programming*. My life was pretty much typography.

TeX leads to a new way of programming.

Literate programming, in my mind, was the greatest spin-off of the TeX project. I learned a new way to program. I love programming, but I really love literate programming. The idea of literate programming is that I'm writing a program for a human being to read rather than a computer to read. It's still a program and it's still doing the stuff, but I'm a teacher to a person. I'm addressing my program to a thinking being, but I'm also being exact enough

so that a computer can understand it as well. Now I can't imagine trying to write a program any other way.

As I'm writing *The Art of Computer Programming*, I realized the key to good exposition is to say everything twice: informally and formally. The reader gets to lodge it in his brain in two different ways, and they reinforce each other. In writing a computer program, it's also natural to say everything in the program twice. You say it in English, what the goals of this part of the program are, but then you say it in your computer language. You alternate between the informal and the formal. Literate programming enforces this idea.

In the comments you also explain what doesn't work, or any subtleties. You can say, "Now note the following. Here is the tricky part in line 5, and it works because of this." You can explain all of the things that a maintainer needs to know. All this goes in as part of the literate program, and makes the program easier to debug, easier to maintain, and better in quality.

After TeX, Don gets to go back to mathematics.

We finished the TeX project; the climax was in 1986. After a sabbatical in Boston I came back to Stanford and plunged into what I consider my main life's work: analysis of algorithms. That's a very mathematical thing, and so instead of having font design visitors to my project, I had great algorithmic analysts visiting my project. I started working on some powerful mathematical approaches to analysis of algorithms that were unheard of in the 1960s when I started the field. Here

"At age 55 I became 'Professor Emeritus of The Art of Computer Programming,' with a capital 'T.' I love that title."

I am in math mode, and thriving on the beauties of this subject.

One of the problems out there that was fascinating is the study of random graphs. Graphs are one of the main focuses of Volume 4, all the combinatorial algorithms, because they're ubiquitous in applications.

Frustrated with the rate of progress, he "retires" to devote himself to "The Art."

I wasn't really as happy as I let on. I mean, I was certainly enjoying the research I was doing, but I wasn't making any progress at all on Volume 4. I'm doing this work on random graphs, and I'm learning all of these things. But at the end of the year, how much more had been done? I've still got 11 feet of preprints stacked up in my closet that I haven't touched, because I had to put that all on hold for the TeX project. I figured the thing that I'm going to be able to do best for the world is finishing *The Art of Computer Programming*.

The only way to do it was to stop being a professor full time. I really had to be a writer full time. So, at age 55 I became "Professor Emeritus of The Art of Computer Programming," with a capital "T." I love that title.

Don is a master at straddling the path between engineering and science.

I always thought that the best way to sum up my professional work is that it has been an almost equal mix of theory and practice. The theory I do gives me the vocabulary and the ways to do practical things that can make giant steps instead of small steps when I'm doing a practical problem. The practice I do makes me able to consider better and more robust theories, theories that are richer than if they're just purely inspired by other theories. There's this symbiotic relationship between those things. At least four times in my life when I was asked to give a kind of philosophical talk about the way I look at my professional work, the title was "Theory and Practice." My main message to the theorists is, "Your life is only half there unless you also get nurtured by practical work."

Software is hard. My experience with TeX taught me to have much more admiration for colleagues that are devot-

ing most of their life to software than I had previously done, because I didn't realize how much more bandwidth of my brain was being taken up by that work than it was when I was doing just theoretical work.

Computers aren't everything: religion is part of his life, too.

I think computer science is wonderful, but it's not everything. Throughout my life I've been in a very loving religious community. I appreciate Luther as a theologian who said you don't have to close your mind. You keep questioning. You never know the answer. You don't just blindly believe something.

I'm a scientist, but on Sundays I would study with other people of our church on aspects of the Bible. I got this strange idea that maybe I could study the Bible the way a scientist would do it, by using random sampling. The rule I decided on was we were going to study Chapter 3, Verse 16 of every book of the Bible.

This idea of sampling turned out to be a good time-efficient way to get into a complicated subject. I actually got too confident that I knew much more than I actually had any right to, because I'm only studying less than 1/500th of the Bible. But a classical definition of a liberal education is that you know everything about something and something about everything.^a

On his working style...

I enjoy working with collaborators, but I don't think they enjoy working with me, because I'm very unreliable. I march to my own drummer, and I can't be counted on to meet deadlines because I always underestimate things. I'm not a great coworker, and I'm very bad at delegating.

I have no good way to work with somebody else on tasks that I can do myself. It's a huge skill that I lack. With the TeX project I think it was important, however, that I didn't delegate the writing of the code. I needed to be the programmer on the first-generation project, and I needed to write the manual, too. If I delegated that, I wouldn't have realized some parts

^a See *3:16 Bible Texts Illuminated*, by Donald Knuth, A-R Editions, 1991.

“I'm worried about the present state of programming. Programmers now are ... supposed to assemble reusable code that somebody else has written... Where's the fun in that? Where's the beauty in that?”

of it are impossible to explain. I just changed them as I wrote the manual.

What is the future of programming?

A program I read when I was in my first year of programming was the SOAP II assembler by Stan Poley at IBM. It was a symphony. It was smooth. Every line of code did two things. It was like seeing a grand master playing chess. That's the first time I got a turn-on saying, “You can write a beautiful program.” It had an important effect on my life.

I'm worried about the present state of programming. Programmers now are supposed to mostly just use libraries. Programmers aren't allowed to do their own thing from scratch anymore. They're supposed to assemble reusable code that somebody else has written. There's a bunch of things on the menu and you choose from these and put them together. Where's the fun in that? Where's the beauty in that? We have to figure out a way we can make programming interesting for the next generation of programmers.

What about the future of science and engineering generally?

Knowledge in the world is exploding. Up until this point we had subjects, and a person would identify themselves with what I call the vertices of a graph. One vertex would be mathematics. Another vertex would be biology.

Another vertex would be computer science, a new one. There would be a physics vertex, and so on. People identified themselves as vertices, because these were the specialties. You could live in that vertex, and you would be able to understand most of the lectures that were given by your colleagues.

Knowledge is growing to the point where nobody can say they know all of mathematics, certainly. But there's so much interdisciplinary work now. We see that a mathematician can study the printing industry, and some of the ideas of dynamic programming apply to book publishing. Wow! There are interactions galore wherever you look. My model of the future is that people won't identify themselves with vertices, but rather with edges—with the connections between. Each person is a bridge between two other areas, and they identify themselves by the two subspecialties that they have a talent for.

Finally, we always ask for life advice.

When I was working on typography, it wasn't fashionable for a computer science professor to do typography, but I thought it was important and a beautiful subject. Other people later told me that they're so glad I put a few years into it, because it made it academically respectable, and now they could work on it themselves. They were afraid to do it themselves. When my books came out, they weren't copies of any other books. They always were something that hadn't been fashionable to do, but they corresponded to my own perception of what ought to be done.

Don't just do trendy stuff. If something is really popular, I tend to think: back off. I tell myself and my students to go with your own aesthetics, what you think is important. Don't do what you think other people think you want to do, but what you really want to do yourself. That's been a guiding heuristic for me all the way through.

And it should for the rest of us. Thank you, Don.

Edited by **Len Shustek**, Chair, Computer History Museum, Mountain View, CA.

© 2008 ACM 0001-0782/08/0800 \$5.00

ACM, Uniting the World's Computing Professionals, Researchers, Educators, and Students

Dear Colleague,

At a time when computing is at the center of the growing demand for technology jobs worldwide, ACM is continuing its work on initiatives to help computing professionals stay competitive in the global community. ACM delivers resources that advance computing as a science and profession.

As a member of ACM, you join nearly 90,000 other computing professionals and students worldwide to define the largest educational, scientific, and professional computing society. Whether you are pursuing scholarly research, building systems and applications, or managing computing projects, ACM offers opportunities to advance your interests.

MEMBER BENEFITS INCLUDE:

- A subscription to the completely redefined **Communications of the ACM**, ACM's flagship monthly magazine
- The option to subscribe to the full **ACM Digital Library**, with improved search functionalities and **Author Profile Pages** for almost every author in computing
- The **Guide to Computing Literature**, with over one million bibliographic citations
- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full and unlimited access to over 3,000 online courses** from SkillSoft
- **Full and unlimited access to 1,100 online books**, featuring 500 from Books24x7®, and 600 from Safari® Books Online, including leading publishers such as O'Reilly (Professional Members only)
- The option to connect with the **best thinkers in computing** by joining **34 Special Interest Groups** or **hundreds of local chapters**
- **ACM's 40+ journals and magazines** at special member-only rates
- **TechNews**, ACM's tri-weekly email digest delivering stories on the latest IT news
- **CareerNews**, ACM's bi-monthly email digest providing career-related topics
- **MemberNet**, ACM's e-newsletter, covering ACM people and activities
- **Email forwarding service & filtering service**, providing members with a free acm.org email address and high-quality **Postini spam filtering**
- And much, much more!

ACM's worldwide network ranges from students to seasoned professionals and includes many of the leaders in the field. ACM members get access to this network, and enjoy the advantages that come from sharing in their collective expertise, all of which serves to keep our members at the forefront of the technology world.

I invite you to share the value of ACM membership with your colleagues and peers who are not yet members, and I hope you will encourage them to join and become a part of our global community.

Thank you for your membership in ACM.

Sincerely,



John R. White
Executive Director and Chief Executive Officer
Association for Computing Machinery



Association for
Computing Machinery

Advancing Computing as a Science & Profession



Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & digital library order form

Priority Code: ACACM29

You can join ACM in several easy ways:

Online http://www.acm.org/join	Phone +1-800-342-6626 (US & Canada) +1-212-626-0500 (Global)	Fax +1-212-944-1318
--	---	-------------------------------

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:

<http://www.acm.org/membership/L2-3/>

Special rates for members of sister societies:

<http://www.acm.org/membership/dues.html>

Please print clearly

Name _____

Address _____

City _____ State/Province _____ Postal code/Zip _____

Country _____ E-mail address _____

Area code & Daytime phone _____ Fax _____ Member number, if applicable _____

Purposes of ACM

ACM is dedicated to:

- 1) advancing the art, science, engineering, and application of information technology
- 2) fostering the open interchange of information to serve both professionals and the public
- 3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature _____

ACM Code of Ethics:

<http://www.acm.org/serving/ethics.html>

choose one membership option:

PROFESSIONAL MEMBERSHIP:

- ACM Professional Membership: \$99 USD
- ACM Professional Membership plus the ACM Digital Library: \$198 USD (\$99 dues + \$99 DL)
- ACM Digital Library: \$99 USD (must be an ACM member)

STUDENT MEMBERSHIP:

- ACM Student Membership: \$19 USD
- ACM Student Membership plus the ACM Digital Library: \$42 USD
- ACM Student Membership PLUS Print CACM Magazine: \$42 USD
- ACM Student Membership w/Digital Library PLUS Print CACM Magazine: \$62 USD

All new ACM members will receive an
ACM membership card.
For more information, please visit us at www.acm.org

Professional membership dues include \$40 toward a subscription to *Communications of the ACM*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

Visa/MasterCard American Express Check/money order

Professional Member Dues (\$99 or \$198) \$ _____

ACM Digital Library (\$99) \$ _____

Student Member Dues (\$19, \$42, or \$62) \$ _____

Total Amount Due \$ _____

Card # _____ Expiration date _____

Signature _____

Online games and virtual worlds have familiar scaling requirements, but don't be fooled: Everything you know is wrong.

BY JIM WALDO

Scaling in Games and Virtual Worlds

I USED TO be like you.

I used to be a systems programmer, working on infrastructure used by banks, telecom companies, and other engineers. I worked on operating systems. I worked on distributed middleware. I worked on programming languages. I wrote tools. I did all of the things that hardcore systems programmers do.

And I knew the rules. I knew that throughput was the real test of scaling. I knew that data had to be kept consistent and durable, and that relational databases are the way to ensure atomicity, and that loss of information is never an option. I knew that clients were getting thinner as the layers of servers increased, and that the best client would be one that contained the least amount of state and allowed the important computations to go on inside the computing cloud. I knew that support for legacy code is vital to the

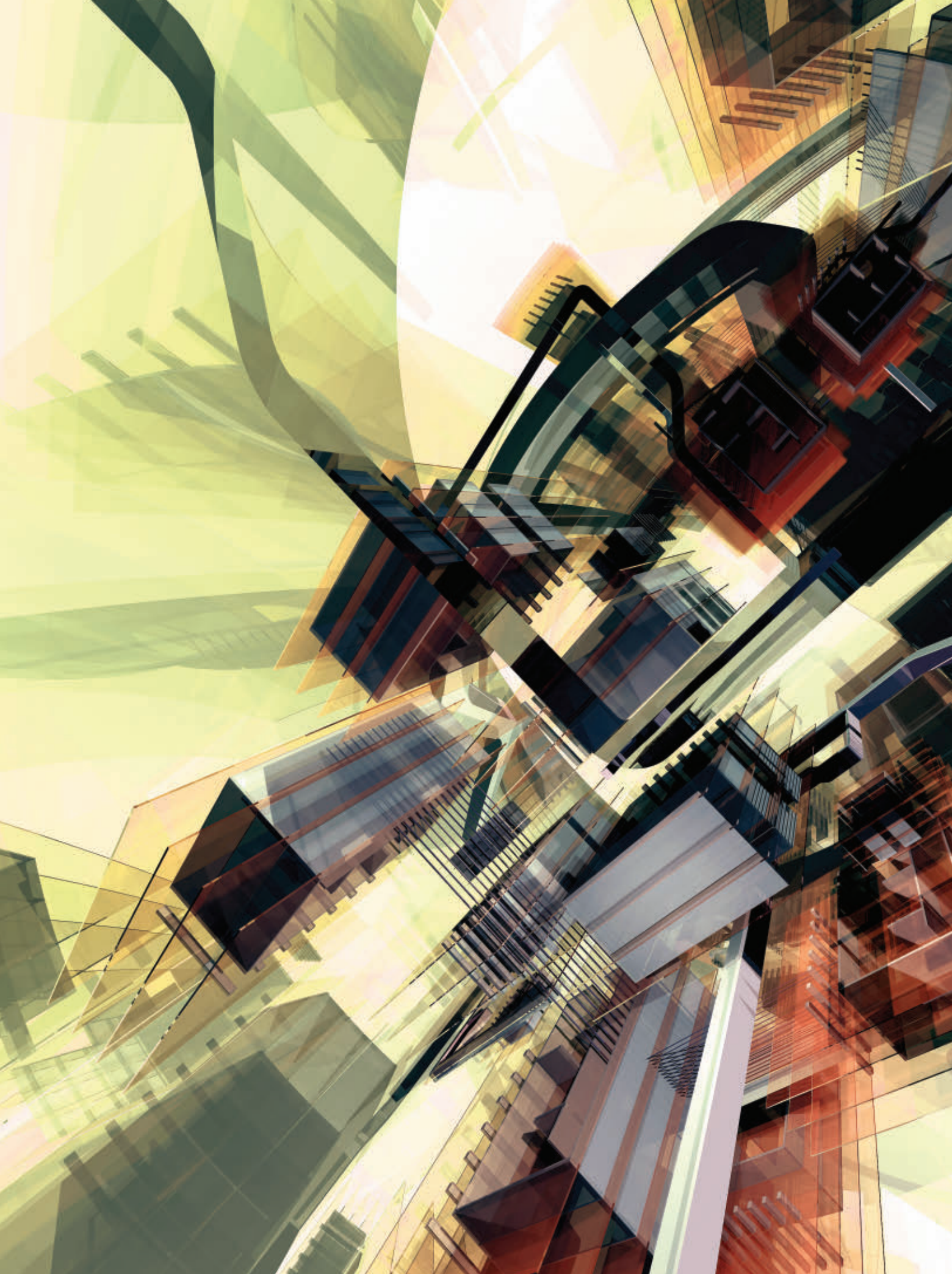
adoption of any new technology, and that most legacy code has yet to be written.

But two years ago my world changed. I was asked to take on the technical architect position on Project Darkstar, a distributed infrastructure targeted to the massive-multiplayer online game and virtual-world market. At first, it seemed like a familiar system. The goal was to scale flexibly by enabling the dynamic addition (or subtraction) of machines to match load. There was a persistence layer and a communication layer. We also wanted to make the programming model as simple as possible, while enabling the system to use all the power of the new generations of multicore chips that Sun (and others) were producing. These were all problems that I had encountered before, so how hard could these particular versions of the problems for this particular market be? I agreed to spend a couple of months on the project, cleaning up the architecture and making sure it was on the right track while I thought about new research topics that I might want to tackle.

The three months have turned into two years (and counting). I've found lots of new research challenges, but they all have to do with finding ways to make the environment for online games and virtual worlds scale. In the process, I have been introduced to a different world of computing, with different problems, different assumptions, and a different environment. At times I feel like an anthropologist who has discovered a new civilization. I'm still learning about the culture and practice of games, and it is a different world.

Everything You Know is Wrong

To understand this new world, the first thing to realize is that it is part of the entertainment industry. Because of this, the most important goal for a game or virtual world is that it be fun. Everything else is secondary to this prime directive. Being fun is not an objective measure, but the goal is to provide an immersive, all-consuming experience



that rewards the player for playing well, is easy to learn but hard to master, and will keep the player coming back again and again.

Most online games center around a story and a world, and the richness of the story and the world has much to do with the success of the game. The design of the game centers on the story and the gameplay. Design of the code that is used to implement the game comes quite a bit later (and is often considered much less interesting). A producer heads the team that builds the game or world. Members of the team include writers, artists, and musicians, as well as coders. The group with the least influence on the game is the coders; their job is to bring the vision of others to reality.

The computational environment for online games or virtual worlds is close to the exact inverse of that found in most markets serviced by the high-tech industry. The clients are anything but thin; game players will be using the highest-end computing platform they can get, or a game console that has been specially designed for the computational rigors of these games.



These client machines will have as much memory as can be jammed into the box, the latest and fastest CPU, and a graphics subsystem that has supercomputing abilities on its own. These clients will also have considerable capacity for persistent storage, since one of the basic approaches to these games is to put as much information as pos-

sible on the client.

The need for a heavyweight client is, in part, an outcome of the evolution of these games. Online games have developed from standalone products, in which everything was done on the local machines. This is more than entropy in the industry, however; keeping as much as possible on the client allows the communication with the server to be minimized, both in the number of calls made to the server and in the amount of information conveyed in those calls. This communication minimization is required to meet the prime directive of fun, since it is part of the way in which latency is minimized in these games.

Latency is the enemy of fun—and therefore the enemy of online games and virtual worlds. This is especially interesting in the case of online games, where the latency of the connection between the client and the servers cannot be controlled. Therefore, the communication protocol needs to be as simple as possible, and the information transmitted from the client to the server must fit into a single packet whenever possible. Further, the server must be designed so that it is doing very little,

ensuring that whatever it is doing can be done very quickly so a response can be sent back to the player. Some interesting tricks have been developed to mask unavoidable latency from the player. These include techniques such as showing prerecorded clips during the loading of a mission or showing a “best guess” immediately at the result

of an action and then repairing any differences between that guess and the actual result when the server responds.

The role of the server is twofold. The most obvious is to allow players to interact with each other in the context of the game. This role is becoming more important and more complex as these games and worlds become more and more elaborate. The original role of the server was to allow players to compete with each other in the game. Now games and worlds are developing their own societies, where players may compete but may also cooperate or simply interact in various ways. Virtual worlds allow users to try out new personalities. Games let players cooperate to do tasks that they would be unable to complete individually. In both, players are finding that a major draw of the technology is using it to connect to other people.

The second role of the server is to be the arbiter of truth between the clients. Whether the client is running on a console or on a personal computer, control rests in the hands of the player. This means the player has access to the client program, and the competitive nature of the games gives the player motivation to alter the client in the player's favor. Even in virtual worlds, where there is only social competition, the desire to enhance the opportunity of the individual player (more commonly known as “cheating”) is common. This requires that the server, which is the one component that is not in control of the players, be the arbiter of the true state of the game. The game server is used both to discourage cheating (by making it much more difficult) and to detect cheating (by seeing patterns of divergence between the game state reported by the client and the game state held by the server). Peer-to-peer technologies might seem a natural fit for the first role of the game server, but this second role means that few if any games or worlds trust their peers enough to avoid the server component.

Current Scaling Strategies


The use of the singular term *server* in the previous section represents a conceptual illusion of the system structure that can be maintained only by the clients of the game or world. In fact, any online game or virtual world will in-

volve a large number of servers (or will have failed so miserably that no one either can or wants to remember the game or world). Using multiple servers is a basic mechanism for scaling the server component of a game to the levels that are being seen in the online world today. World of Warcraft has reported more than five million subscribers with hundreds of thousands active at any one time. Second Life reports usage within an order of magnitude of World of Warcraft, and there is some evidence that sites such as Webkinz or Club Penguin are even more popular. A single server is not able to handle such load, no matter how efficient the representation. Even if a single server could deal with this load, such a server would be far too expensive for the smaller loads that are encountered (sometimes by the same games or worlds) at times of low demand (or in parts of the product's life cycle when demand has decreased).


Having multiple servers means that part of building the game is deciding how to partition the load over these servers. Two techniques are commonly used in both online games and virtual worlds. Sometimes only one of the two techniques is used, sometimes both, depending on the nature of the game or world.

The first technique is to exploit the geography of the game or world, decomposing the game into different areas, each of which can be mapped to a hosting server. For example, an island in Second Life corresponds to a physical server running the code for the shared reality of the world. Similarly, different areas of the World of Warcraft universe are hosted on different physical machines. Anyone who is in the area will connect to the same server, and interactions among the players on that server can be localized (and optimized). Actions happening in a different part of the world are not likely to affect those in this part of the world, so the communication traffic between servers can be kept small.

The second technique is known as *sharding*. A shard is a copy of a part of the game or virtual world. Different shards reside on different servers, and players who are assigned to one shard can interact with the world and other players in the shard, but will not see (or



With the possible exception of the highest end of scientific computing, no other kind of software has ridden the advances of Moore's Law as aggressively as game or virtual-world programs.



be able to interact with) players or objects in other shards. Shards not only allow more players to be supported in the world, but also permit independent explorations into the world by different sets of players. Thus, when a new quest or mission is added to a game, it will often be replicated with multiple shards so that more than one player (or group of players) can experience the quest or mission in its original state.

Although sharding and geographic decomposition allow multiple servers to be used to handle the load on a single game or world, they do present the developer with significant challenges. By creating noninteracting copies of parts of a world, shards isolate the players in different shards from each other. This means that players who want to share their experience of the world or game need to become aware of the different shards that are being offered, and arrange to be placed in the same shard. As the number of players who want to be in the same shard increases (some guilds, or groups of players who cooperatively play in a single game over an extended period of time, have hundreds of members), the difficulty of coordinating placement into shards increases and interferes with the experience of the world. While shards allow scale, they do so at the price of player interaction.

Geographic decomposition does not limit player interaction, but does require that the designers of the game be able to predict the size of a geographic area that will be the correct unit of decomposition. If one geographic area becomes very popular, play on that area will slow down as the server associated with the area is overloaded. If a geographic area is less popular than originally predicted, computer hardware (and money) will be wasted on that section because not enough players are there. Since the geographic decomposition is hardwired into the code of the game or world, changing the decomposition in response to observed user behavior requires rewriting part of the game or world itself. This takes time, can introduce bugs, and is very costly. While this is being done, gameplay can be adversely affected. In extreme cases, this can have a major financial impact. When World of Warcraft was in-

troduced, the demand for the game so outstripped the capacity that had been built into the game that subscriptions had to be closed off for months while the code that distributed the game was rewritten.


Changing Chip Architectures

Scaling over a set of machines is a distributed computing problem, and the game and virtual-world programming culture has had little experience with this set of problems. This is hardly the only place where scaling requires the game programmer to learn a new set of skills. A change in the trend of chip design also means these programmers must learn skills they have never had to exercise before.


With the possible exception of the highest end of scientific computing, no other kind of software has ridden the advances of Moore's Law as aggressively as game or virtual-world programs. As chips have gotten faster, games and virtual worlds have become more realistic, more complex, and more immersive. Serious gamers invest in the very best equipment that they can obtain, and then use techniques such as overclocking to push even more performance out of those systems.

Now, however, chip designers have decided to exploit Moore's Law in a different way. Rather than increasing the speed of a chip, they are adding multiple cores to a chip running at the same (or sometimes slower) clock speed. There are many good reasons for this, from simplified design to lower power consumption and heat production, but it means that the performance of a single program will not automatically increase when you run the program on a new chip. Overall performance of a group of programs may increase (since they can all run in parallel) but not the single program (unless it can be broken into multiple, cooperating threads). Games are written as single-threaded programs, however.

In fact, games and virtual worlds (and especially the server side of these programs) should be perfect vehicles to show the performance gains possible with multicore chips and groups of cooperating servers. Games and virtual worlds are embarrassingly parallel, in that most of what goes on in them is independent of the other things that



By backing the data in a persistent fashion rather than keeping it in main memory, we gain some inherent reliability that has not been exhibited by games or worlds in the past.



are happening. Of the hundreds of thousands of players who are active in World of Warcraft at any one time, only a very small number will be interacting with any particular player. The same is true in Second Life and nearly all large-scale games or worlds.

The problem is that the culture that has grown up around games and virtual worlds is not one that understands or is overly familiar with the programming techniques that are required to exploit the parallelism inherent in these systems. These are people who grew up on a single (PC) machine, running a single thread. Asking them to master the intricacies of concurrent programming or distributed systems takes them away from their concentration on the game or world experience itself. Even when they have the desire, they don't have the time or the experience to exploit these new technologies.

Project Darkstar

It is for these reasons that we started Project Darkstar (<http://www.projectdarkstar.com>), a research effort attempting to build a server-side infrastructure that will exploit the multithreaded, multicore chips being produced and scaled over a large group of machines while presenting the programmer with the illusion that he or she is developing in a single-threaded, single-machine environment. Hiding threading and distribution is, in the general case, probably not a good idea (see <http://research.sun.com/techrep/1994/abstract-29.html> for a full argument). Game and world servers tend to follow a very restricted programming model, however, in which we believe we can hide both concurrency and distribution.

The model is a simple event-based one in which input from the client is received by the server, which then sets off a task in response to that event. These tasks can change the state of the world (by moving a player, changing the state of an object, or the like) and initiate communication. The communication can be to a single client or to a group of clients that are all subscribed to the same communication channel.

We chose this model largely because this is the way most game and virtual-world servers are already structured. The challenge was then to keep this

model and allow servers written in this style to be scaled over multiple cores (running multiple threads) and multiple servers. We were not trying to take existing code and allow it to run within our system. This would have made the task much more difficult and would not have corresponded to the realities of the game and virtual-world culture. Game and world servers are written from scratch for each game or world, perhaps reusing some libraries but rarely, once running, being rehosted into a different environment. Efforts to bring different platforms into the game are restricted to the client side, where new consoles bringing in new players may be worth the effort.

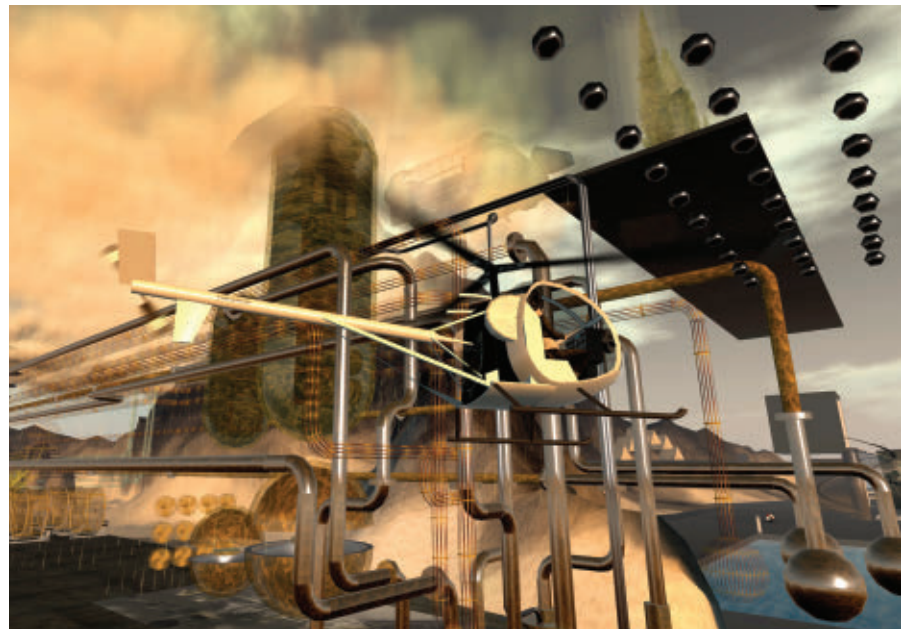
Darkstar provides a container in which the server runs. The container provides interfaces to a set of services that allow the game server to keep persistent state, establish connections with clients, and construct publish/subscribe channels with sets of clients. Multiple copies of the game server code can run in multiple instances of the Darkstar container. Each copy can be written as if it was the only one active (and, in fact, it may be the only one active for small-scale games or worlds). Each of the servers is structured as an event loop—the main loop listens on a session with a client that is established when the client logs in. When a message is delivered, the event loop is called. The loop can then decode the message and determine the game or world action that is the appropriate response. It then dispatches a task within the container.

Each of these tasks can read or change data in the world through the Darkstar data service, communicate with the client, or send messages to groups of other game or world participants via a channel. Under the covers, the task is wrapped in a transaction. The transaction is used to ensure that no conflicting concurrent access to the world data will occur. If a task tries to change data that is being changed by some other concurrent task, the data service will detect that conflict. In that case, one of the conflicting tasks will be aborted and rescheduled; the other task should run to completion. Thus, when the aborted task is retried, the conflict should have disappeared and the task should run to completion.

This mechanism for concurrency control does require that all tasks access all of their data through the Darkstar data service. This is a departure from the usual way of programming game or world servers, where data is kept in memory to decrease latency. By using results from the past 20 years of database research, we believe that we can keep the penalty for accessing through a data service small by caching data in intelligent ways. We also believe that by using the inherent parallelism in these games, we can increase the overall performance of the game as the number of players increases, even if there is a small penalty for individual data access. Our data store is not based on a standard SQL database since we don't need the full functionality such databases provide. What we need is something that gives us fast access to persistently stored objects that can be identified in simple ways. Our current implementation uses the Berkeley Database for this, although we have abstracted our access to it to provide the opportunity to use other persistence layers if required.

crash can cause the loss of any change in the game or world since the last time the system was checkpointed. This can sometimes be hours of play, which can cause considerable consternation among the customers and expensive calls to the service lines. By keeping all data persistently, we believe we can ensure that no more than a few seconds of game or world interaction will be lost in the case of a server crash. In the best case, such a crash won't even be noticed by the players, as the tasks that were on the server will be transferred to another server in a fashion that is transparent to the player.

The biggest payoff for requiring that all data be kept in the data store is that it helps to make the tasks that are generated by the response to events in the game portable. Since the data store can be accessed by any of a cluster of machines that are running the Darkstar stack and the game logic, there is no data that cannot be moved from machine to machine. We do the same with the communication mechanisms, ensuring that a session or channel that is connecting the game and some set of



Concurrency control is not the only reason to require that all data be accessed through the data store. By backing the data in a persistent fashion rather than keeping it in main memory, we gain some inherent reliability that has not been exhibited by games or worlds in the past. Storing all of the data in memory means that a server

clients is abstracted through the Darkstar stack. This allows us to move the task using the session or channel to another machine without affecting the semantics of the task talking over the session or channel.

This task portability means we can dynamically balance the load on a set of machines running the game or virtual

world. Rather than splitting the game up into regions or shards at compile time, virtual worlds or games based on the Darkstar stack can move load around the network of server machines at runtime. While the participant might see a short increase in latency during the move, the overall latency will be decreased after the move. By moving tasks, we not only can balance the load on the machines involved, but also try to collocate tasks that are accessing the same set of data or that are communicating with each other. All of these mechanisms allow us to determine, while the game is being played, which tasks (and which users) should be placed on the same server.

The project is in its early stages of development and deployment. It is based on an open-source licensing model and community, so we are relying on our users to educate us about the needs of the community that will build the games and worlds that will run on the infrastructure. The research is part computer science and part anthropology, but each of the cultures has an opportunity to learn much from the other.

about the underlying concurrency of the system. The most obvious of these is in the design of the data structures. One of the earliest users of our code was getting terrible performance from the system. When we looked at the code, we discovered that a single object was written to on every task, updating a global piece of game state. By designing the server in this way, this user effectively serialized all of the tasks that were running in the system, making it impossible for the server to get any advantage from the inherent parallelism in the game. Some minor redesign, breaking the single object into many (much smaller) objects, removed this particular bottleneck, with resulting gains in overall performance. This experience also taught us that we need to educate users of the system in the design of independent data structures that can be accessed in parallel.

Nor has our own implementation been without some excitement. When we moved from a multithreaded server that ran on a single machine to an implementation that runs on multiple machines, we expected some degradation in the performance of the single-

multiple machines is greater than that on a single machine, and discovering and recovering from such contention takes longer. We are working on removing the choke points so that adding equipment actually adds capacity.

Measuring the performance of the system is made especially challenging by the lack of any clear notion of what the requirements of the target servers are. Game developers are notoriously secretive, and the notion of a characteristic load for a game or virtual world is not something that is well documented. We have some examples that have been written by the team or by people we know in the game world, but we cannot be sure that these are accurate reflections of what is being written by the industry. Our hope is that the open-source community that is beginning to form around the project will aid in the production of useful performance and stress tests.

Seen in a broader light, the project has and continues to be an interesting experiment in building levels of abstraction for the world of multithreaded, distributed systems. The problems we are tackling are not new. Large Web-serving farms have many of the same problems with highly variable demand. Scientific grids have similar problems of scaling over multiple machines. Search grids have similar issues in dealing with large-scale environments solving embarrassingly but not completely parallel problems.

What makes online games and virtual worlds interestingly different are the very different requirements they bring to the table compared to these other domains. The interactive, low-latency environment is very different from grids, Web services, or search. The growth from the entertainment industry makes the engineering disciplines far different from those others, as well. Solving these problems in this new environment is challenging, and adds to our general knowledge of how to write software for the emerging class of multithreaded, multicore, distributed systems.

And best of all, it's fun. □

Jim Waldo is a Distinguished Engineer with Sun Microsystems Laboratories, Burlington, MA, where he conducts research on large-scale distributed systems.

© 2008 ACM 0001-0782/08/0800 \$5.00



Even at this early stage, it is clear that this is going to be a complex venture. While early experience with the code has shown that the programming model does relieve the game or world server programmer from thinking about threads and locking, it has also shown that there are places where they do have to understand something

machine system. We were delighted to find that the single-node system degradation was not nearly as large as we thought it would be, but we found that additional machines lowered the capacity of the overall system. When presented with these measurements, this was not all that surprising to understand—the possibility for contention on mul-

Leaders in the storage world offer valuable advice for making more effective architecture and technology decisions.

BY MACHE CREEGER, MODERATOR

CTO Storage Roundtable

FEATURING SEVEN WORLD-CLASS storage experts, this roundtable discussion is the first in a new series of CTO Forums focusing on the near-term challenges and opportunities facing the commercial computing community. Overseen by the ACM Professions Board, the goal of this series is to provide working IT managers expert advice so they can make better decisions when investing in new architectures and technologies. This is the first installment of the

discussion, with a second installment slated for publication in the September issue.

Recognizing that Usenix and ACM serve similar constituencies, Ellie Young, Usenix Executive Director, graciously invited us to hold our panel during the Usenix Conference on File and Storage Technologies (FAST '08) in San Jose, Feb. 27, 2008. Ellie and her staff were extremely helpful in supporting us during the conference and all of us at ACM greatly appreciate their efforts.

—*Stephen Bourne*

Participants

Steve Kleiman—Senior Vice President and Chief Scientist, Network Appliances.

Eric Brewer—Professor, Computer Science Division, University of California, Berkeley, Inktomi co-founder (acquired by Yahoo).

Erik Riedel—Head, Interfaces & Architecture Department, Seagate Research, Seagate Technology.

Margo Seltzer—Herchel Smith Professor of Computer Science, Professor in the Division of Engineering and Applied Sciences, Harvard University, Sleepycat Software founder (acquired by Oracle Corporation), architect at Oracle Corporation.

Greg Ganger—Professor Electrical and Computer Engineering, School of Computer Science, Director, Parallel Data Lab, Carnegie Mellon University.

Mary Baker—Research Scientist, HP Labs, Hewlett-Packard.

Kirk McKusick—Past president, Usenix Association, BSD and FreeBSD architect.

Moderator

Mache Creeger—Principal, Emergent Technology Associates.

MACHE CREEGER: Welcome to you all. Today we're talking about storage issues that are specific to what people are coming into contact with now and what they can expect in the near term. Why don't we start with energy consumption and see where that takes us?

ERIC BREWER: Recently I decided to rebuild my Microsoft Windows XP PC from scratch and for the first time tried to use a 32GB flash card instead of a hard drive. I'm already using network-attached storage for everything important and information on local disk is

easily re-created from the distribution CD. Flash consumes less energy and is much quieter.

Although this seemed like a good idea, it didn't work out that well because XP apparently does a great deal of writing to its C drive during boot. Writing to flash is not a good idea, as the device is limited in the number and bandwidth of writes. Even though the read time for flash is great, I found the boot time on the Windows machine to be remarkably poor. It was slower than the drive I was replacing and I'm go-

ing to have to go back to a disk in my system. But I still like the idea and feel that the thing that I need to boot my PC should be a low-power flash device with around 32GB of storage.

ERIK RIEDEL: This highlights one of the problems with the adoption of new technologies. Until the software is appropriately modified to match the new hardware, you don't get the full benefit. Much of the software we run today is old. It was designed for certain paradigms, certain sets of hardware, and as we move to new hardware the old software doesn't match up.

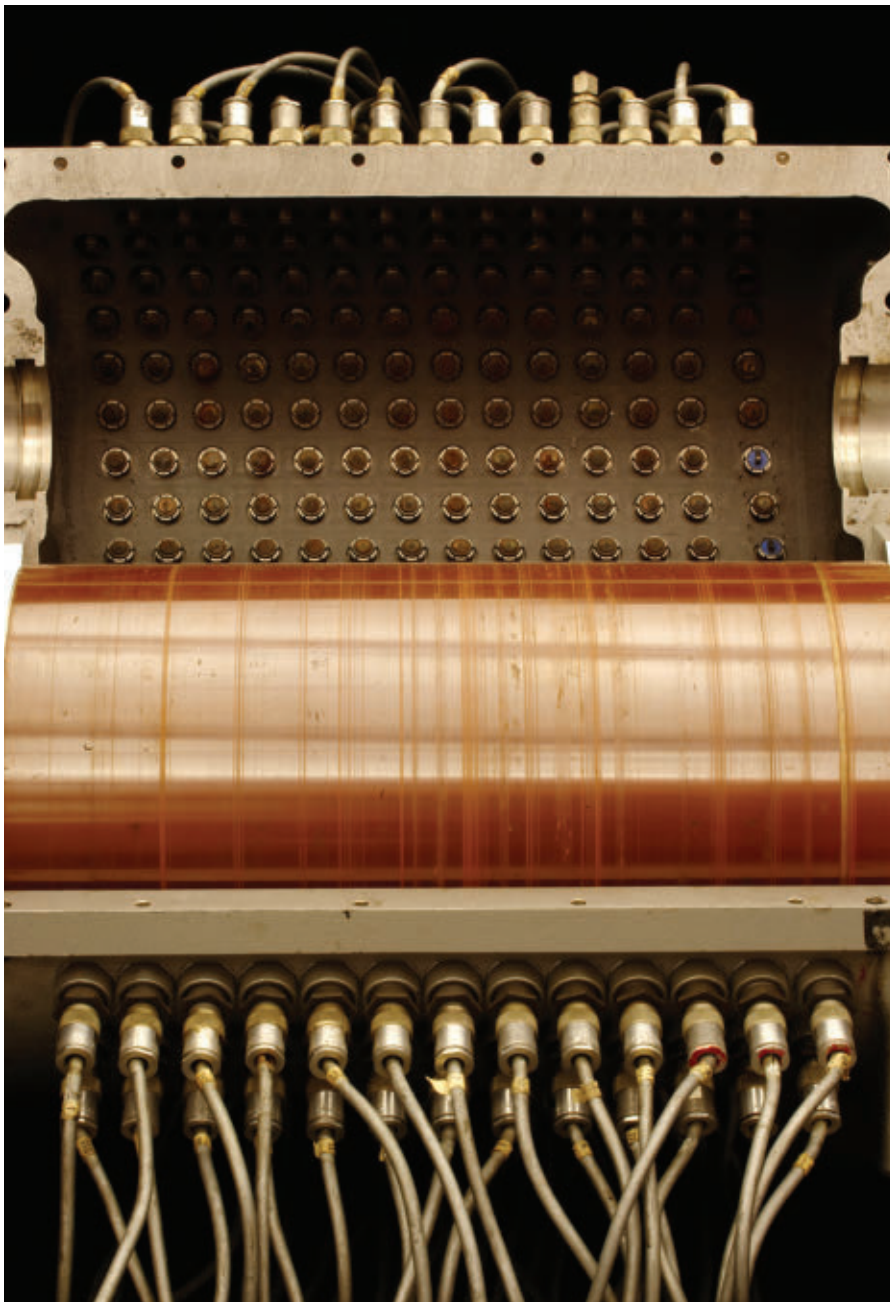
MACHE CREEGER: I've had a similar experience. In my house, my family has gotten addicted to MythTV—a free, open source, client-server, DVR (Digital Video Recorder) that runs on Linux (<http://www.mythtv.org/>). Mindful of energy consumption, I wanted to get rid of as many disk drives as possible. I first tried to go diskless and do a network boot of my clients off of the server. I found it awfully difficult to get a network-booted Linux client to be configured the way I wanted. Things like NFS did not come easily and you had to do a custom kernel if you wanted to include stuff outside a small standard set.

Since I wanted small footprint client machines, and was concerned about heat and noise, I took a look at flash, but quickly noted that it was write-limited. Because I did not have a good handle on my outbound writes, flash didn't seem to be a particularly good candidate for my needs.

I settled on laptop drives, which seemed to be the best compromise. Laptop drives have lots of storage, are relatively cheap, can be shaken, don't generate a lot of heat, and do not require a lot of power to operate. For small audiovisual client computers, laptop drives seem to be the state-of-the-art right solution for me right now.

ERIK RIEDEL: Seagate has been selling drives specifically optimized for DVRs. The problem is we don't sell them to the retail channel, but to integrators like TIVO and Comcast. Initially, the optimization was for sound. We slowed down the disk seek times and did other things with the materials to eliminate the clicky-clacky sound.

Recently, power is more of a concern. You have to balance power with storage capacity. When you go to a



Developed in the 1950s, magnetic drums were the first mechanical “direct access” storage devices. Made of a nickel-cobalt substrate coated with powdered iron, data was recorded by magnetizing small surface regions organized into long tracks of bits.

notebook drive, it's a smaller drive with smaller platter, so there are fewer bits. For most DVRs, you still care about how many HD shows you can put on it (a typical hour of high-definition TV uses over five times the storage capacity of standard definition TV).

MARY BAKER: Talk about noise. We have three TBs of storage at home. What used to be my linen closet is now the machine room. While storage appliances are supposed to be happy sitting in a standard home environment, with three of them, I get overheating failures. Our house isn't air conditioned, but the linen closet is. It doesn't matter how quiet the storage is because the air conditioner is really loud.

MACHE CREEGER: What we're finding in this little microcosm are the trade-offs that people need to consider. The home server is becoming a piece of house infrastructure for which people have to deal with issues of power, heat generation, and noise.

KIRK MCKUSICK: We have seven machines in our house and we wanted to cut our power consumption at 59 cents a kilowatt-hour. We got Soekris boxes that will support either flash or laptop drives (<http://www.soekris.com/>). The box uses six watts plus the power consumption of the attached storage device.

The first machine we tried was our FreeBSD gateway. We used flash and it worked out great. FreeBSD doesn't write anything until after it's gone multi-user and as a result we were able to configure our gateway to be almost write-free.

Armed with our initial success, we focused on our Web server. We discovered the Web server, Apache, writes stuff all the time and our first flash device write-failed after 18 months. But flash technology seems to be improving. After we replaced it with a 2X-sized device, it has not been as severely impacted by writes. The replacement has been going strong for almost three years.

MARGO SELTZER: My guys who are studying flash claim that the write problem is going to be a thing of the past very soon.

STEVE KLEIMAN: Yes and no. Write limits are going to go down over time. However, as long as capacity increases enough so that at a given write rate you're not using it up too fast, it's okay. It is correct to think of flash as a con-



STEVE KLEIMAN

The implications of flash are profound. I've done the arithmetic. For as long as I can remember it's been about a 100-to-1 ratio between main memory and disk in terms of dollars per gigabyte. Flash sits right in the middle.



sumable, and you have to organize your systems that way.

KIRK MCKUSICK: But disks are also consumable, they only last three years.

STEVE KLEIMAN: Disks are absolutely consumable. They are also obsolete after five years, as you don't want to use the same amount of power to spin something that's a quarter of the storage space of the current technology.

The implications of flash are profound. I've done the arithmetic. For as long as I can remember it's been about a 100-to-1 ratio between main memory and disk in terms of dollars per gigabyte. Flash sits right in the middle. In fact, if you look at the projections, at least on a raw cost basis, by 2011–2012 flash will overlap high-performance disk drives in terms of dollars per gigabyte.

Yet flash has two orders of magnitude better dollars per random I/O operation than disk drives. Disk drives have a 100-to-1 difference in bandwidth between random and serial access patterns. In flash that's not true. It's probably a 2- or 3-to-1 difference between read and write, but the dynamic range is much less.

GREG GANGER: It's much more like RAM in that way.

STEVE KLEIMAN: Yes. My theory is that whether it's flash, phase-change memory, or something else, there is a new place in the memory hierarchy. There was a big blank space for decades that is now filled and a lot of things that need to be rethought. There are many implications to this, and we're just beginning to see the tip of the iceberg.

MARY BAKER: There are a lot of people who agree with you, and it's going to be fun to watch over the next few years. There is the JouleSort contest (<http://joulesort.stanford.edu/>) to see, within certain constraints—performance or size—what is the lowest power at which you can sort a specific data set. The people who have won so far have been experimenting with flash.

STEVE KLEIMAN: I went to this Web site that ranked the largest databases in the world. I think the largest OLTP (Online Transaction Processing) databases were between 3TB–10TB. I know from my friends at Oracle that if you cache 3% to 5% of an OLTP database, you're getting a lot of the interesting stuff. What that means is a few thousand dollars worth of flash can cache the largest

OLTP working set known today. You don't need hundreds of thousands of dollars of enterprise "who-ha" if a few thousand dollars will do it.

With companies like Teradata and Netezza you have to ask if doing all these things to reorganize the data for DSS (Decision Support Systems) is even necessary anymore?

MACHE CREEGER: For the poor IT managers out in Des Moines struggling to get more out of their existing IT infrastructure, you're saying that they should really look at existing vendors that supply flash caches?

STEVE KLEIMAN: No. I actually think that flash caches are a temporary solution. If you think about the problem, caches are great with disks because there is a benefit to aggregation. If I have a lot of disks on the network, I can get a better level of performance than I could from my own single disk dedicated to me because I have more arms working for me.

With DRAM-based caches, I get a benefit to aggregation because DRAM is so expensive it's hard to dedicate it to any single node. Neither of these is true of network-based flash caches. You can only get a fraction of performance of flash by sticking it out over the network. I think flash migrates to both sides, to the host and to the storage system. It doesn't exist by itself in the network.

MACHE CREEGER: Are there products or architectures that people can take advantage of?

STEVE KLEIMAN: Sure. I think for the next few years, cache will be an important thing. It's an easy way to do things. Put some SSDs (Solid State Disks) into some of the caching products, or arrays, that people have and it's easy. There'll be a lot of people consuming SSDs. I'm just talking about the long term.

MACHE CREEGER: This increases performance overall, but what about the other issue: power consumption?

STEVE KLEIMAN: I'm a power consumption skeptic. People do all these architectures to power things down, but the lowest-power disk is the one you don't own. Better you should get things into their most compressed form. What we've seen is that if you can remove all the copies that are out in the storage system and make it only one instance, you can eliminate a lot of storage that

you would otherwise have to power. When there are hundreds of copies of the same set of executables, that's a lot of savings.

MARGO SELTZER: You're absolutely right, getting rid of duplication helps reduce power. But that's not inconsistent; it's a different kind of power management. If you look at the cost of storage it's not just the initial cost, but also the long-term cost, such as management and power. Power is a huge fraction, and de-duplication is one way to cut that down. Any kind of lower-power device, of which flash memory is one example, is going to be increasingly more attractive to people as power becomes increasingly more expensive.

STEVE KLEIMAN: I agree. Flash can handle a lot of the very expensive, high-power workloads—the heavy random I/Os. But I am working on the assumption that disks still exist. On a dollar-per-gigabyte basis, there's at least a 5-to-1 ratio between flash and disks, long term.

MARGO SELTZER: If it costs five times more to buy a flash disk than a spinning disk, how long do I have to use a flash disk before I've made up that 5X cost in power savings over spinning disk?

STEVE KLEIMAN: It's a fair point. Flash consumes very little power when you are not accessing it. Given the way electricity costs are rising, the cost of power and cooling over a five-year life for even a "fat" drive can approach the raw cost of the drive. That's still not 5X. The disk folks are working on lower-power operating and idle modes that can cut the power by half or more without adding more than a few seconds latency to access. So that improves things to only 50% over the raw cost of the drive.

Look at tape-based migration systems. The penalty for making a bad decision is really bad, because you have to go find a tape, stick it in the drive, and wait a minute or two. Spinning up a disk or set of disks is almost the same since it can take longer than 30 seconds. Generally those tape systems were successful where it was expected behavior that the time to first data access might be a minute. Obviously, the classic example is backup and restore, and that's where we see spin-down mostly used today.

If you want to apply these ideas to

general-purpose, so-called "unstructured" data, where it's difficult to let people know that accessing this particular data set might have a significant delay, it's hard to get good results. By the time the required disks have all spun up, the person who tried to access an old project file or follow a search hit is on the phone to IT. With the lower-power operating modes, the time to first access is reasonable and the power savings is significant. By the way, much of the growth in data over the past few years has been in unstructured data.

ERIK RIEDEL: That's where the key solutions are going to come from. Look at what the EPA is doing with their recent proposals for Energy Star in the data center. They address a whole series of areas where you need to think about power. They have a section about the power management features you have in your device. The way that it's likely to be written is you can get an Energy Star label if you do two of the following five things, choosing between things like de-duplication, thin provisioning, or spin-down.

But if you look at the core part of the spec, there's a section where they're focused on idle power. Idle power is where we have a big problem in storage. The CPU folks can idle the CPU. If there is nothing to do then it goes idle. The problem is storage systems still have to store the data and be responsive when a data request comes in. That means time-to-data and time-to-ready are important. In those cases people really do need to know about their data. The best idle power for storage systems is to turn the whole thing off, but that doesn't give people access to their data.

We've never been really careful because we haven't had to be. You could just keep spending the watts and throwing in more equipment. When you start asking "What data am I actually using and how am I using it?" you have to do prediction.

STEVE KLEIMAN: My point is that there is so much low-hanging fruit with de-duplication, compression, and lower-power operating modes before you have to turn the disk off that we can spend the next four or five years just doing that and save much more energy than spinning it down will do.

ERIK RIEDEL: We are going to have to

know more about the data and the applications. Look at the history of an earlier technology we all know about: RAID. There are multiple reasons to do RAID. You do it for availability, to protect the data, and for performance benefits. There are also areas where RAID does not provide any benefits. When we ask our customers why they are doing RAID, nobody knows which of the benefits are more important to them.

We've spent all this time sending them to training classes, teaching them about the various RAID-levels, and how you calculate the XORs. What they know is if they want to protect their data, they've got to turn it up to RAID5, and if they've got money lying around, they want to turn it up to RAID10. They don't know why they're doing that, they're just saying, "This is what I'm supposed to do so I'll do it." There isn't the deeper understanding of how the data and applications are being used. The model is not there.

MARGO SELTZER: I don't think that's going to change. We're going to have to figure out the RAID equivalent for power management because I don't think people are going to figure out their data that way. It's not something that people know or understand.

KIRK MCKUSICK: Or they're going to put flash in front of the disk, so you can have the disk power down. You can dump it into flash and then update the disk when it becomes available.

ERIC BREWER: Many disks have some NVRAM (Non-Volatile RAM) in them anyway, so I feel like one could absorb the write burst while the drive wakes up. We should be able to hide that. At least in my consumer case, I know that one disk can handle my read load. Enterprise is a more complicated, but that's a lot of disks we can shut down.

STEVE KLEIMAN: I disagree. Flash caches can help with a lot of applications being consumed in the enterprise. However, because there is a 10-to-1 cost factor, there are areas where flash adds no benefit. You have to let the disk show through so that cache misses are addressed. That is very hard to predict.

We've long passed the point where you can delete something. Typically, you don't know what is important and what is not and you can't spend the time and money to figure it out. So



MACHE CREEGER

A mantra that I learned early on in databases was more spindles are better. What you're all saying now is that we have to challenge that. More spindles are better, but at what cost?



you end up keeping everything, which means in some sense everything's equally valued. The problem is that you need a certain level of minimum reliability or redundancy into all the data because it's hard to distinguish what is important and what's not. It's not just RAID. People are going to want to have disaster recovery strategy. They're not going to have just one copy of this thing, RAID or no RAID.

ERIK RIEDEL: At a recent event in my department to discuss storage power, we had a vendor presentation that showed a CPU scaling system. When system administrators feel they are getting close to peak power they can access a master console and turn back all the processors by 20%. That's a system that they have live running today. And they do it without fear. They figure that applications are balanced and somehow all the applications—the Web servers, the database servers—will adjust to everything running 20% slower.

When our group saw that, it became clear that we are going to have to figure out what the equivalent of that is for storage. We need to be able to architect storage systems so that an administrator has the option of saying, "I need it to consume 20% or 30% less power for the next couple of hours."

MACHE CREEGER: A mantra that I learned early on in databases was more spindles are better. More spindles allow you to have more parallelism and a wider data path. What you're all saying now is that we have to challenge that. More spindles are better, but at what cost? Yes, I can run a database on one spindle, but it's not going to be a particularly responsive one. It won't have all the performance of a 10-spindle database, but it's going to be cheaper to run.

STEVE KLEIMAN: If you think about the database example, I don't know about that. You can put most of the working set on flash. You don't have to worry about spinning it.

MARGO SELTZER: That's the key insight here. Flash has two attractive properties: It handles random I/O load really well and it's also very power efficient. I think you have to look at how that's going to play into the storage hierarchy and how it's going to help.

In some cases you may be using flash as a performance enhancer, as a power enhancer, or both. This gets back to

Erik's point, which is that today people don't know why they're using RAID. It may very well be the same with flash.

GREG GANGER: The general model of search engines is you want to have a certain cluster that handles a given load. When you want to increase the load you can handle, you essentially replicate that entire cluster. It's the unit of replication that makes management easier.

When it's Christmas Eve and the service load is low, you could actually power down many of the replicas. While I do not believe this has been done yet, it seems like the thing to do as power costs continue to be a larger issue. In these systems there is already a great degree of replication in order to provide more spindles during high-load periods.

MACHE CREEGER: You all said that there is low-hanging fruit to take advantage of. Are there things you can do today as profound as server virtualization?

STEVE KLEIMAN: The companion to

server virtualization is storage virtualization. Things like snapshots and clones take whole golden images of what you're going to run and instantaneously make a copy so that only the parts that have changed are additional. You might have 100 virtual servers out there with what they think are 100 images, but it's only one golden image and the differences. That's an amazing savings. It's the same thing that's going on with server virtualization; it's almost the mirror image of it.

What has come about over the last few years is the ability to share the infrastructure. You may have one infrastructure, but it's still a hundred different images, you're actually not sharing the data. That's changed in the last five years since we have had cloning technology. This allows you to get this tremendous so-called thin-provisioning savings.

ERIC BREWER: I disagree with something said earlier, which is that it's becoming hard to delete stuff. I feel that deletion is a fundamental human right

because it gets to the core of what is private and what rights you have over data about you. I want to be able to delete my own stuff, but I also want to be able to delete from groups that have data about me that I no longer trust. A lot of this is a legal issue, but I hate to feel like the technical things are going to push us away from the ability to delete.

STEVE KLEIMAN: That's a good point. While it's hard to expend the intellectual effort to decide what you want to delete, once you've expended that effort, you should be able to delete. The truth is that it's incredibly hard to delete something. Not only do you have to deal with the disks themselves, but also the bits that are resident on the disk after you "delete" them, and the copies, and the backups on tape.

One of the things that is part of our product right now, and which we continue to work on, is the ability to fine-grain encrypt information and then throw away the key. That deletes the information itself, the copies of the in-



Invented by IBM in 1956, the first Model 350 disk drive contained 50 24-inch diameter disks and stored a total of 5MB. IBM later added removable disk platters to its drives; these platters provided archival data storage.

formation, and the copies of the information on tape.

MARGO SELTZER: It seems that there are two sides to this. I agree that's a nice solution to the deletion problem, but it concerns me because you may get the unintended consequence, which is now you've got a key management problem. Given my own ability to keep track of my passwords, the thought of putting stuff I care about on an encrypted device where if I lose the key, I've lost my data forever, is a little scary.

STEVE KLEIMAN: We have a technology that does exactly that. It turns into a hierarchical key management system. Margo's right. When you care about doing stuff like that, you have to get serious about it. Once you lose or delete that key, it's really, really, truly, gone.

MARGO SELTZER: And given that my greatest love of snapshots comes from that time that I inadvertently deleted the thing that I didn't want to, inadvertent key deletion really scares me.

STEVE KLEIMAN: That's why people won't do it, right? I think it'll be done for very specific reasons with pre-thought intent that says, "Look, for legal reasons, because I don't want to be sued, I don't want this document to exist after five years."

Today, data ownership has a very real burden. For example, you have an obligation to protect things like your customers' credit card numbers, or Social Security numbers, and this obligation has a real cost. This gives you a way of relieving yourself of that burden when you want to.

MARGO SELTZER: I hear you and I believe it at one level, but at another level, I can't help but think of the dialogue boxes that pop up that say, "Do you really mean to do this?" and we're all trained to click on them and say "Yes." I'm concerned about how seriously humans will take an absolute delete.

ERIK RIEDEL: Margo, you've pointed out a much bigger problem. Today, one of the key problems within all security technology is that the usability is essentially zero. With regards to Web page security, it's amazing what people are willing to click and ignore. As long as there's a lock icon somewhere on the page, it's fine.

ERIC BREWER: If we made deletion a



MARGO SELTZER

Given my own ability to keep track of my passwords, the thought of putting stuff I care about on an encrypted device where if I lose the key, I've lost my data forever, is a little scary.



right, this would get sorted out. I could expect business relationships of mine to delete all records about me after our relationship ceased. The industry would figure it out. If you project out 30 years, the amount you can infer given what's out there is much worse than what's known about you today.

MARY BAKER: It's overwhelming and there's no way to pull it back in. Once it's out there, there's no control.

MACHE CREEGER: Now that we all agree that there should be a way to make information have some sort of time-to-live or be able to disappear at some future direction, what recommendations can we make?

MARGO SELTZER: There's a fundamental conflict here. We know how to do real deletion using encryption, but for every benefit there's a cost. As an industry, people have already demonstrated that the cost for security is too high. Why are our systems insecure? No one is willing to pay the cost in either usability or performance to have true security.

In terms of deletion, there's a similar cost-benefit relationship. There is a way to provide the benefit, but the cost in terms of risk of losing data forever is so high that there's a tension. This fundamental tension is never going to be fully resolved unless we come up with a different technology.

ERIC BREWER: If what you want is time to change your mind, we could just wait awhile to throw away the key.

MARGO SELTZER: The best approach I've heard is that you throw away bits of the key over time. Throwing away one bit of the key allows recovery with a little bit of effort. Throw away the second bit and it becomes harder, and so on.

ERIC BREWER: But ultimately you're either going to be able to make it go away or you're not. You have to be willing to live with what it means to delete. Experience always tells us that there's regret when you delete something you would rather keep. □

Mache Creeger (mache@creeger.com) is a technology industry veteran based in Silicon Valley. Along with being a columnist for *ACM Queue*, he is the principal of Emergent Technology Associates, marketing and business development consultants to technology companies worldwide.

© 2008 ACM 0001-0782/08/0800 \$5.00



DOI:10.1145/1378704.1378718

**There's a lot we can learn
from CORBA's mistakes.**

BY MICHI HENNING

The Rise and Fall of CORBA

DEPENDING ON EXACTLY when one starts counting, CORBA (Common Object Request Broker Architecture) is around 15 years old. During its lifetime, CORBA has moved from being a bleeding-edge technology for early adopters, to being a popular middleware, to being a niche technology that exists in relative obscurity. It is instructive to examine why CORBA—despite once being heralded as the “next-generation technology for e-commerce”—suffered this fate. CORBA’s history is one that the computing industry has seen many times, and it seems likely that current middleware efforts, specifically Web services, will reenact a similar history.

In the early 1990s, persuading programs on different machines to talk to each other was a nightmare, especially if different hardware, operating systems, and programming languages were involved: programmers either used sockets and wrote an entire protocol stack themselves or their programs didn’t talk at all. (Other early middleware, such as Sun ONC, Apollo NCS, and DCE, was tied to C and

Unix and not suitable for heterogeneous environments.)

After a false start with CORBA 1.0, which was not interoperable and provided only a C mapping, the OMG (Object Management Group) published CORBA 2.0 in 1997. It provided a standardized protocol and a C++ language mapping, with a Java language mapping following in 1998. This gave developers a tool that allowed them to build heterogeneous distributed applications with relative ease. CORBA rapidly gained popularity and quite a number of mission-critical applications were built with the technology. CORBA’s future looked rosy.

During CORBA’s growth phase in the mid- and late 1990s, major changes affected the computing landscape, most notably, the advent of Java and the Web. CORBA provided a Java language mapping, but it did nothing to cooperate with the rapidly expanding Web. Instead of waiting for CORBA to deliver a solution, companies turned to other technologies and started building their e-commerce infrastructures based on Web browsers, HTTP, Java, and EJB (Enterprise Java Beans).

In addition, developers who had gained experience with CORBA found that writing any nontrivial CORBA application was surprisingly difficult. Many of the APIs were complex, inconsistent, and downright arcane, forcing the developer to take care of a lot of detail. In contrast, the simplicity of component models, such as EJB, made programming a lot simpler (if less flexible), so calls for a CORBA component model became louder and louder. A component model was a long time in coming, however. Work was started in 1996 on a CBOF (Common Business Object Facility), but that effort got bogged down in political infighting and was eventually abandoned, to be replaced by the CCM (CORBA Component Model). A specification for CCM was finally published in late 1999, but was largely a nonevent:

► The specification was large and complex and much of it had never been implemented, not even as a proof of concept. Reading the document made

it clear that CCM was technically immature: sections of it were essentially unimplementable or, if they were implementable, did not provide portability.

- ▶ No commercial CORBA vendor committed to implement CCM.

- ▶ Even if implementations had been available by the time CCM was published, it was too late. EJB had become entrenched in the industry to the point where another component technology had no chance of success.

CMM failure did little to boost the confidence of CORBA customers, who were stuck with complex technology.

Meanwhile, the industry's need for middleware was stronger than ever. After some experience with e-commerce systems that used HTTP, HTML, and the CGI (Common Gateway Interface), it had become clear that building distributed systems in this way had serious limitations. Without a proper type system, applications were reduced to parsing HTML to extract semantics, which amounted to little more than screen-scraping. The resulting systems turned out to be very brittle. On the other hand, EJB had a proper type system but was limited to Java and so not suited for many situations. There were also a few flies in the CORBA ointment:

- ▶ Commercial CORBA implementations typically cost several thousand dollars per development seat, plus, in many cases, runtime royalties for each deployed copy of an application. This limited broader acceptance of the platform—for many potential customers, CORBA was simply too expensive.

- ▶ The platform had a steep learning curve and was complex and hard to use correctly, leading to long development times and high defect rates. Early implementations also were often riddled with bugs and suffered from a lack of quality documentation. Companies found it difficult to find the expert CORBA programmers they needed.

Microsoft never embraced CORBA and instead chose to push its own DCOM (Distributed Component Object Model). This kept much of the market either sitting on the fence or using DCOM instead, but DCOM could not win the middleware battle either, because it worked only on Windows. (A port of DCOM to Unix by Software AG never gained traction.) Microsoft eventually dropped DCOM after several

failed attempts to make it scale. By that time, the middleware market was in a very fragmented state, with multiple technologies competing, but none was able to capture sufficient mindshare to unify distributed systems development.

Another important factor in CORBA's decline was XML. During the late 1990s, XML had become the new silver bullet of the computing industry: almost by definition, if it was XML, it was good. After giving up on DCOM, Microsoft wasn't going to leave the worldwide e-commerce market to its competitors and, rather than fight a battle it could not win, it used XML to create an entirely new battlefield. In late 1999, the industry saw the publication of SOAP (Simple Object Access Protocol). Originally developed by Microsoft and DevelopMentor, and then passed to W3C for standardization, SOAP used XML as the on-the-wire encoding for remote procedure calls.

SOAP had serious technical shortcomings, but, as a market strategy, it was a masterstroke. It caused further fragmentation as numerous vendors clambered for a share of the pie and moved their efforts away from CORBA and toward the burgeoning Web services market. For customers, this added more uncertainty about CORBA's viability, often prompting them to put investment in the technology on hold.

CORBA suffered another blow when the Internet bubble burst in early 2001. The industry's financial collapse drove many software companies out of the market and forced the survivors to refocus their efforts. The result was significant attrition in the number of commercial CORBA products. Before the collapse, several vendors had already dropped or deemphasized their CORBA products and, after the collapse, more followed. What in the mid- to late 1990s had been a booming market with many competing products had suddenly turned into a fringe market with far fewer vendors, customers, and investment. By then, open source implementations of CORBA were available that partially compensated for the departure of the commercial vendors, but this was not enough to recover the lost mindshare and restore the market's confidence: CORBA was no longer the darling child of the industry.

Today, CORBA is used mostly to wire together components that run inside companies' networks, where commu-

nication is protected from the outside world by a firewall. It is also used for real-time and embedded systems development, a sector in which CORBA is actually growing. Overall, however, CORBA's use is in decline and it cannot be called anything but a niche technology now.

Given that only a few years ago, CORBA was considered to be the cutting edge of middleware that promised to revolutionize e-commerce, it is surprising to see how quickly the technology was marginalized, and it is instructive to examine some of the deeper reasons for the decline.

Technical Issues

Obviously, a number of external factors contributed to the fall of CORBA, such as the bursting of the Internet bubble and the competition with other technologies, such as DCOM, EJB, and Web services. One can also argue that CORBA was a victim of industry trends and fashion. In the computing industry, the technical excellence of a particular technology frequently has little to do with its success—mindshare and marketing can be more important factors. These arguments cannot fully account for CORBA's loss of popularity, however. After all, if the technology had been as compelling as was originally envisaged, it is unlikely that customers would have dropped it in favor of alternatives.

Technical excellence is not a sufficient prerequisite for success but, in the long term, it is a necessary prerequisite. No matter how much industry hype might be pushing it, if a technology has serious technical shortcomings, it will eventually be abandoned. This is where we can find the main reasons for CORBA's failure.

Complexity: The most obvious technical problem is CORBA's complexity—specifically, the complexity of its APIs. Many of CORBA's APIs are far larger than necessary. For example, CORBA's object adapter requires more than 200 lines of interface definitions, even though the same functionality can be provided in about 30 lines—the other 170 lines contribute nothing to functionality, but severely complicate program interactions with the CORBA runtime.

Another problem area is the C++ language mapping. The mapping is difficult to use correctly and contains many

pitfalls that lead to bugs, particularly with respect to thread safety, exception safety, and memory management. A number of other examples of overly complex and poorly designed APIs can be found in the CORBA specification, such as the naming, trading, and notification services, all of which provide APIs that are error-prone and difficult to use. Also, CCM configuration is so complex it cannot be used productively without employing additional tool support.

Poorly designed interfaces and language mappings are a very visible part of any technology because they are the “coal face” of software development: they are the point at which developers and the platform meet, and their usability and safety has major impact on development time and defect count. Obviously, any technology that suffers from endemic complexity does little to endear itself to developers, and does even less to endear itself to managers.

Complexity also arises from architectural choices. For example, CORBA’s IORs (interoperable object references) are opaque entities whose contents are to remain hidden from developers. This is unfortunate for three reasons:


- ▶ Opaque references pretty much force the use of a naming service because clients cannot create object references without the help of an external service. This not only complicates system development and deployment, but also introduces redundant state into the system (with the concomitant risk of corrupting that state) and creates an additional failure point.

- ▶ Opaque references considerably complicate a number of APIs. For example, CORBA’s interceptor APIs would be far simpler had object references been made transparent.


- ▶ Opaque references require remote calls to compare object identity reliably. For some applications, the overhead of these calls is prohibitive.

Another source of complexity is the type system. For example, CORBA’s interface definition language provides a large set of types, among them unsigned integers, fixed-point and extended-precision floating-point numbers, bounded and unbounded sequences, as well as arrays, and an “Any” type that can store values of arbitrary type.

Supporting these types complicates many APIs (in particular, the interfaces



Today, CORBA is used mostly to wire together components that run inside companies’ networks, where communication is protected from the outside world by a firewall. It is also used for real-time and embedded systems development, a sector in which CORBA is actually growing. Overall, however, CORBA’s use is in decline and it cannot be called anything but a niche technology now.



for introspection and dynamic invocation) and leads to subtle portability problems. For example, Java does not support unsigned types, so use of an unsigned integer in an interface can lead to overflow problems when a Java client communicates with a C++ server. Similarly, on platforms without native support for fixed-point or double-precision floating-point numbers, implementations must emulate these types.

Emulations are difficult to implement such that they behave identically across platforms, and they require additional APIs. This adds further complexity and is a source of hard-to-diagnose interoperability problems.

Finally, some of the OMG’s early object services specifications, such as the life cycle, query, concurrency control, relationship, and collection services, were not only complex, but also performed no useful function whatsoever. They only added noise to an already complex suite of specifications, confused customers, and reinforced CORBA’s reputation of being hard to use.

Insufficient Features: CORBA provides quite rich functionality, but fails to provide two core features:

- Security.* CORBA’s unencrypted traffic is subject to eavesdropping and man-in-the-middle attacks, and it requires a port to be opened in the corporate firewall for each service. This conflicts with the reality of corporate security policies. (Incidentally, this CORBA shortcoming was major factor in the rise of SOAP. Not having to open a port in the corporate firewall and sending everything via port 80 was seen as a major advantage, despite the naïvety of that idea.) The OMG made several attempts at specifying security and firewall traversal for CORBA, but they were abandoned as a result of technical shortcomings and lack of interest from firewall vendors.

- Versioning.* Deployed commercial software requires middleware that allows for gradual upgrades of the software in a backward-compatible way. CORBA does not provide any such versioning mechanism (other than versioning by derivation, which is utterly inadequate). Instead, versioning a CORBA application generally breaks the on-the-wire contract between client and server. This forces all parts of a deployed application to be replaced at once, which is typically infeasible. (This shortcoming

of CORBA was another major factor in the rise of SOAP. The supposedly loosely coupled nature of XML was seen as addressing the problem, despite this idea being just as naive as funneling all communications through port 80.)

For a commercial e-commerce infrastructure, lack of security and versioning are quite simply showstoppers—many potential e-commerce customers rejected CORBA for these reasons alone.

A number of other technical issues plague CORBA, among them:

- ▶ Design flaws in CORBA's interoperability protocol make it pretty much impossible to build a high-performance event distribution service.

- ▶ The on-the-wire encoding of CORBA contains a large amount of redundancy, but the protocol does not support compression. This leads to poor performance over wide-area networks.

- ▶ The specification ignores threading almost completely, so threaded applications are inherently nonportable (yet threading is essential for commercial-grade applications).

- ▶ CORBA does not support asynchronous server-side dispatch.

- ▶ No language mappings exist for C# and Visual Basic, and CORBA has completely ignored .NET.

This list of problems is just a sample and could be extended considerably. Such issues affect only a minority of customers, but add to CORBA's bad press and limit its market.

Procedural Issues

Technical problems are at the heart of CORBA's decline, raising the question of how it is possible for a technology produced by the world's largest software consortium to suffer such flaws. As it turns out, the technical problems are a symptom rather than a cause.

The OMG is an organization that publishes technology based on consensus. In essence, members vote to issue an RFP (request for proposals) for a specification, member companies submit draft specifications in response, and the members vote on which draft to accept as a standard. In theory, this democratic process is fair and equitable but, in practice, it does not work:

There are no entry qualifications to participate in the standardization process. Some contributors are experts in the field, but, to be blunt, a large num-

ber of members barely understand the technology they are voting on. This repeatedly has led to the adoption of specifications with serious technical flaws.

RFPs often call for a technology that is unproven. The OMG membership can be divided into roughly two groups: users of the technology and vendors of the technology. Typically, it is the users who would like to expand CORBA to add a capability that solves a particular problem. These users, in the hope that vendors will respond with a solution to their problem, drive issuance of an RFP. Users, however, usually know little about the internals of a CORBA implementation. At best, this leads to RFPs containing requirements that are difficult to implement or have negative performance impact. At worst, it leads to RFPs that are little more than requests for vendors to perform magic. Instead of standardizing best existing practice, such RFPs attempt to innovate without prior practical experience.

Vendors respond to RFPs even when they have known technical flaws. This may seem surprising. After all, why would a vendor propose a standard for something that is known to suffer technical problems? The reason is that vendors compete with each other for customers and are continuously jostling for position. The promise to respond to an RFP, even when it is clear that it contains serious problems, is sometimes used to gain favor (and, hopefully, contracts) with users.

Vendors have a conflict of interest when it comes to standardization. For vendors, standardization is a two-edged sword. On the one hand, standardization is attractive because it makes it easier to sell the technology. On the other hand, too much standardization is seen as detrimental because vendors want to keep control over features that distinguish their product from the competition.

Vendors sometimes attempt to block standardization of anything that would require a change to their existing products. This causes features that should be standardized to remain proprietary or to be too vaguely specified to be useful. Some vendors also neglect to distinguish standard features from proprietary ones, so customers stray into implementation-specific territory without warning. As a result, porting a CORBA application to a different vendor's

implementation can be surprisingly costly; customers often find themselves locked into a particular product despite all the standardization.

RFPs are often answered by several draft specifications. Instead of choosing one of the competing specifications, a common response of OMG members is to ask the submitters to merge their features into a single specification. This practice is a major cause of CORBA's complexity. By combining features, specifications end up as the kitchen sink of every feature thought of by anyone ever. This not only makes the specifications larger and more complex than necessary, but also tends to introduce inconsistencies: different features that, in isolation, are perfectly reasonable can subtly interact with each other and cause semantic conflicts.

Major vendors occasionally stall proceedings unless their pet feature makes it into the merged standard. This causes the technology process to degenerate into political infighting, forces foul compromises, and creates delays. For example, the first attempt at a component model was a victim of such infighting, as was the first attempt at a C++ mapping. Both efforts got bogged down to the point where they had to be abandoned and restarted later.

The OMG does not require a reference implementation for a specification to be adopted. This practice opens the door to castle-in-the-air specifications. On several occasions, the OMG has published standards that turned out to be partly or wholly unimplementable because of serious technical flaws. In other cases, specifications that could be implemented were pragmatically unusable because they imposed unacceptable runtime overhead. Naturally, repeated incidents of this sort are embarrassing and do little to boost customer confidence. A requirement for a reference implementation would have forced submitters to implement their proposals and would have avoided many such incidents.

Overall, the OMG's technology adoption process must be seen as the core reason for CORBA's decline. The process encourages design by committee and political maneuvering to the point where it is difficult to achieve technical mediocrity, let alone technical excellence. Moreover, the addition of disjointed features leads to a gradual ero-

sion of the architectural vision.

CORBA's numerous technical flaws have accumulated to a point where it is very difficult to fix or add anything without breaking something else. For example, every revision of CORBA's interoperability protocol had to make incompatible changes, and many fixes and clarifications to the protocol had to be reworked several times because of unforeseen interactions with features that were added over time.

Can We Learn from the Past?

A democratic process such as the OMG's is uniquely ill suited for creating good software. Despite the known procedural problems, however, the industry prefers to rely on large consortia to produce technology. Web services, the current silver bullet of middleware, uses a process much like the OMG's and, by many accounts, also suffers from infighting, fragmentation, lack of architectural coherence, design by committee, and feature bloat. It seems inevitable that Web services will enact a history quite similar to CORBA's.

What steps should we take to end up with a better standards process and better middleware? Seeing that procedural failures are the root cause of technical failures, I suggest at least the following:


Standards consortia need iron-cast rules to ensure that they standardize existing best practice. There is no room for innovation in standards: throwing in "just that extra little feature" inevitably causes unforeseen technical problems, despite the best intentions.

No standard should be approved without a reference implementation. This provides a first-line sanity check of what is being standardized. (No one is brilliant enough to look at a specification and be certain it does not contain hidden flaws without actually implementing it.)


No standard should be approved without having been used to implement a few projects of realistic complexity. This is necessary to weed out poor APIs: too often, the implementers of an API never actually use their own interfaces, with disastrous consequences for usability.

Interestingly, the open source community has done a much better job of adhering to these rules than have industry consortia.

Open source innovation usually is subject to a Darwinian selection process. Dif-



Technical problems are at the heart of CORBA's decline. This raises the question of how it is possible for a technology that was produced by the world's largest software consortium to suffer such flaws. As it turns out, the technical problems are a symptom rather than a cause.



ferent developers implement their ideas of how something should work, and others try to use the feature and critique or improve it. That way, the software is extensively scrutinized and tested, and only the "fittest" version survives. (Many open source projects formalize this process with alternating experimental and production releases: the experimental releases act as the test bed and evolutionary filter.)

To create quality software, the ability to say "no" is usually far more important than the ability to say "yes." Open source embodies this in something that can be called "benevolent dictatorship": even though many people contribute to the overall effort, a single expert (or a small cabal of experts) ultimately accepts or rejects each proposed change. This preserves the original architectural vision and stops the proverbial too many cooks from spoiling the broth.

At the heart of these open source practices are two essential prerequisites: cooperation and trust. Without cooperation, the evolutionary process cannot work; and without trust, no cabal of experts can act as an ultimate arbiter. This, however, is precisely where software consortia find their doom. It is naïve to put competing vendors and customers into a consortium and expect them to come up with a high-quality product—commercial realities ensure that cooperation and trust are the last things on the participants' minds.

Of course, software consortia contribute to an evolutionary process just as much as open source projects do. But it is the commercial marketplace that acts as the test bed and evolutionary filter, and it is the customers who, with their wallets, act as the (usually not so benevolent) dictator. This amounts to little more than an industry that throws up silver bullets and customers who leap after them like lemmings over a cliff. Until we change this process, the day of universal e-commerce middleware is as far away as ever. ■

Michi Henning (michi@zeroc.com) is chief scientist of ZeroC, Palm Beach Garden, FL. From 1995 to 2002, he worked on CORBA as a member of the OMG's architecture board and as an ORB implementer, consultant, and trainer.

A previous version of this article appeared in the June 2006 issue of *ACM Queue*.

© 2008 ACM 0001-0782/08/0800 \$5.00

DOI:10.1145/1378704.1378719

Data generated as a side effect of game play also solves computational problems and trains AI algorithms.

BY LUIS VON AHN AND LAURA DABBISH

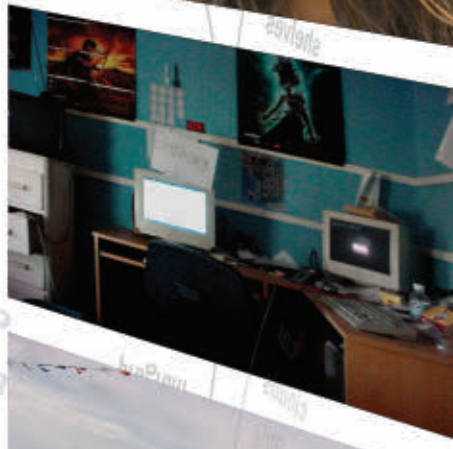
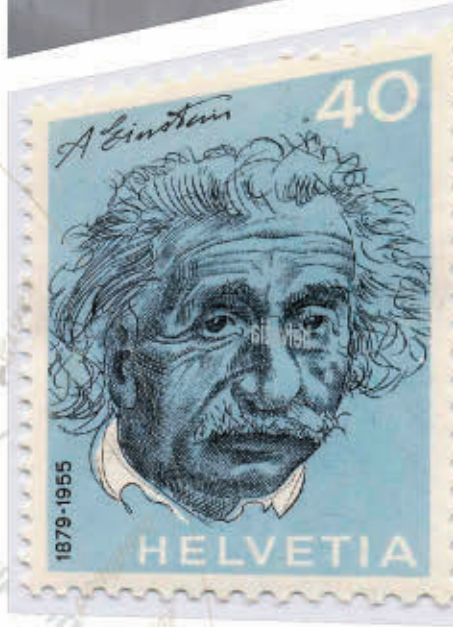
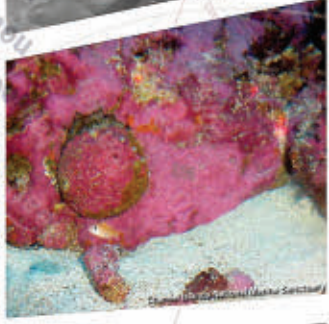
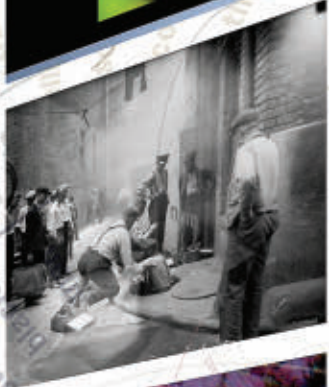
Designing Games With A Purpose

MANY TASKS ARE trivial for humans but continue to challenge even the most sophisticated computer programs. Traditional computational approaches to solving such problems focus on improving artificial-intelligence algorithms. Here, we advocate a different approach: the constructive channeling of human brainpower through computer games. Toward this goal, we present general design principles for the development and evaluation of a class of games we call “games with a purpose,” or GWAPs, in which people, as a side effect of playing, perform tasks computers are unable to perform.

The Entertainment Software Association (www.theesa.com/facts/gamer_data.php) has reported that more than 200 million hours are spent each day playing computer and video games in the U.S. Indeed, by age 21, the average American has spent more than 10,000 hours playing such games¹⁵—equivalent to five years of working a full-time job 40 hours per week.

VISUALIZATION BY BEN FRY





cute

red

animal

ocean

fish

kid

trees

rock

world

dark

dark

branch

branch

wheel

race

red

purple

blue

blur

tire

derby

green

picture

photos

smile

heart

photo

snow

boy

you

woman

kid

green

point

girl

man

music

sea
stone
rock

war
sand
guns
yellow
gun

movie
johnny
depp
fire

clouds
house
green

black
white
cold

green
rock
beach

fight
boot
tank
gun

movie
johnny
depp
fire

cloud
restaurant
green

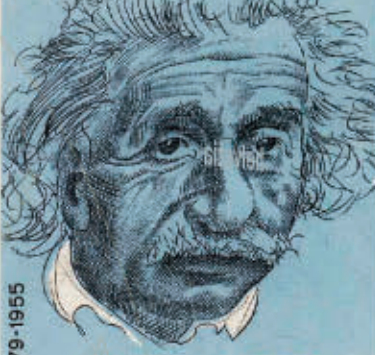
cover
standing
picture
ice
cold
photo

Proof... not promises.
Introducing a scientific breakthrough

prevenge
anti-aging treatment

A Einstein

40



1879-1955

HELVETIA

THE NEW WORLD


avivie

What if this time and energy were also channeled toward solving computational problems and training AI algorithms?


People playing GWAPs²²⁻²⁵ perform basic tasks that cannot be automated. The ESP Game,²² a.k.a. the Google Image Labeler (images.google.com/imagelabeler/), is a GWAP in which people provide meaningful, accurate labels for images on the Web as a side effect of playing the game; for example, an image of a man and a dog is labeled “dog,” “man,” and “pet.” The game is fast-paced, enjoyable, and competitive; as of July 2008, 200,000 players had contributed more than 50 million labels; try it yourself at www.gwap.com. These labels can be used to improve Web-based image search, which typically involves noisy information (such as filenames and adjacent text). Rather than using computer-vision techniques that do not work well enough, the ESP Game constructively channels its players to do the work of labeling images in a form of entertainment.

Other GWAPs include Peekaboom,²⁵ which locates objects within images (and has been played more than 500,000 human-hours); Phetch,²³ which annotates images with descriptive paragraphs; and Verbosity,²⁴ which collects commonsense facts in order to train reasoning algorithms. In each, people play not because they are personally interested in solving an instance of a computational problem but because they wish to be entertained.

The ESP Game, introduced in 2003, and its successors represent the first seamless integration of game play and computation. How can this approach be generalized? Our experience building and testing GWAPs with hundreds of thousands of players has helped us spell out general guidelines for GWAP development. Here, we articulate three GWAP game “templates” representing three general classes of games containing all the GWAPs we’ve created to date. They can be applied to any computational problem to construct a game that encourages players to solve problem instances. Each template defines the basic rules and winning conditions of a game in a way that is in the players’ best interest to perform the intended computation. We also describe a set of design principles that comple-



People play not because they are personally interested in solving an instance of a computational problem but because they wish to be entertained.



ment the basic game templates. While each template specifies the fundamental structure for a class of games, the general design principles make the games more enjoyable while improving the quality of the output produced by players. Finally, we propose a set of metrics defining GWAP success in terms of maximizing the utility obtained per human-hour spent playing the game.

Related Work

Though previous research recognized the utility of human cycles and the motivational power of gamelike interfaces, none successfully combined these concepts into a general method for harnessing human processing skills through computer games.

Networked individuals accomplishing work. Some of the earliest examples of networked individuals accomplishing work online, dating to the 1960s, were open-source software-development projects. These efforts typically involved contributions from hundreds, if not thousands, of programmers worldwide. More recent examples of networked distributed collaboration include Wikipedia, by some measures equal in quality to the *Encyclopaedia Britannica*.⁶

The collaborative effort by large numbers of networked individuals makes it possible to accomplish tasks that would be much more difficult, time consuming, and in some cases nearly impossible for a lone person or for a small group of individuals to do alone. An example is the recent Amazon Mechanical Turk system (developed in 2005, www.mturk.com/mturk/welcome) in which large computational tasks are split into smaller chunks and divvied up among people willing to complete small amounts of work for some minimal amount of money.

Open Mind Initiative. The Open Mind Initiative^{18,19} is a worldwide research endeavor developing “intelligent” software by leveraging human skills to train computers. It collects information from regular Internet users, or Netizens, and feeds it to machine-learning algorithms. Volunteers participate by providing answers to questions computers cannot answer (such as “What is in this image?”), aiming to teach computer programs common-

sense facts. However, the Open Mind approach involves two drawbacks: reliance on the willingness of unpaid volunteers to donate their time and no guarantee that the information they enter is correct. GWAPs differ from Open Mind in that they are designed to be enjoyable while ensuring that the data they collect is free from error.

Interactive machine learning. Another area leveraging human abilities to train computers is “interactive machine learning”⁴ in which a user provides examples to a machine-learning system and is given real-time feedback as to how well an algorithm is learning. Based on the feedback, the user is able to determine what new examples should be given to the program. Some instances of this approach have utilized human perceptual skills to train computer-vision algorithms to recognize specific objects.

Making work fun. Over the past 30 years, human-computer-interaction researchers have recognized and written about the importance of enjoyment and fun in user interfaces.^{16,26} For example, systems (such as the StyleCam) aim to use gamelike interaction to increase enjoyment and engagement with the software.²¹ Many researchers have suggested that incorporating gamelike elements into user interfaces could increase user motivation and the playfulness of work activities.^{16,26} Some projects have taken this notion further, turning the user interface itself into a game. For instance, PSDoom provides a first-person-shooter-style interface for system-administrator-related tasks.^{2,3} The idea of turning work tasks into games is increasingly being applied in children’s learning activities.¹² Researchers note, as we do here, that it is important to not simply slap a game-like interface onto work activities but to integrate the required activities into the game itself; there must be tight interplay between the game interaction and the work to be accomplished.

Desire to Be Entertained

The GWAP approach is characterized by three motivating factors: an increasing proportion of the world’s population has access to the Internet; certain tasks are impossible for computers but easy for humans; and people spend lots of time playing games on computers.

In contrast to other work that has attempted to use distributed collections of individuals to perform tasks, the paradigm we describe here does not rely on altruism or financial incentives to entice people to perform certain actions; rather, they rely on the human desire to be entertained. A GWAP, then, is a game in which the players perform a useful computation as a side effect of enjoyable game play. Every GWAP should be associated with a computational problem and therefore generate an input-output behavior.

A game can be fully specified through a goal players try to achieve (the winning condition) and a set of rules that determines what players can and cannot do during the game. A GWAP’s rules should encourage players to correctly perform the necessary steps to solve the computational problem and, if possible, involve a probabilistic guarantee that the game’s output is correct, even if the players do not want it to be correct.

The key property of games is that people want to play them. We therefore sidestep any philosophical discussions about “fun” and “enjoyable,” defining a game as “successful” if enough human-hours are spent playing it.

We advocate a transformative process whereby a problem is turned into a GWAP. Given a problem that is easy for humans but difficult or impossible for computers, the process of turning the problem into a GWAP consists of first creating a game so that its structure (such as rules and winning condition) encourages computation and correctness of the output. Having created many GWAPs, including the ESP Game, Peekaboom, Phetch, and Verbosity, we explore three game-structure templates that generalize successful instances of human computation games: output-agreement games, inversion-problem games, and input-agreement games.

Output-agreement games. Output-agreement games (see Figure 1) are a generalization of the ESP Game (see the sidebar “The ESP Game and Verbosity” on page 65) to its fundamental input-output behavior:

Initial setup. Two strangers are randomly chosen by the game itself from among all potential players;

Rules. In each round, both are given the same input and must produce out-

Figure 1: In this output-agreement game, players are given the same input and must agree on an appropriate output.

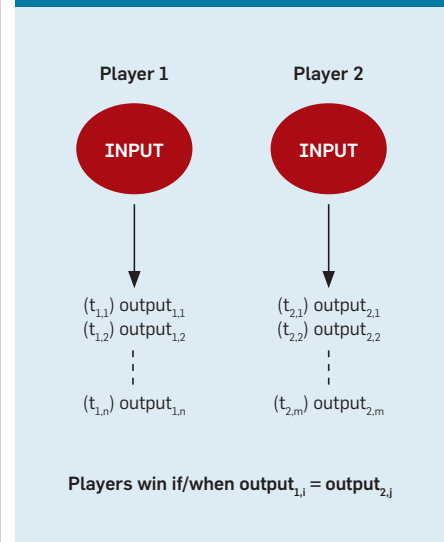
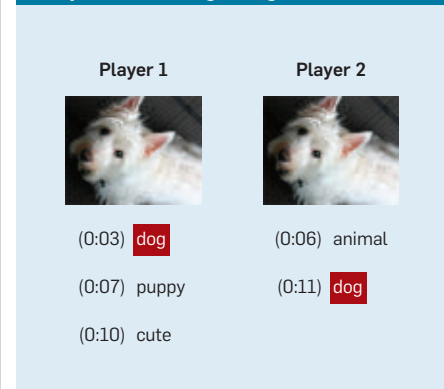


Figure 2: In this output-agreement game, the partners are agreeing on a label.



puts based on the input. Game instructions indicate that players should try to produce the same output as their partners. Players cannot see one another’s outputs or communicate with one another; and

Winning condition. Both players must produce the same output; they do not have to produce it at the same time but must produce it at some point while the input is displayed onscreen.

When the input is an image and the outputs are keyword descriptions of the image, this template becomes the ESP Game (see Figure 2).

Since the two players cannot communicate and know nothing about each other, the easiest way for both to produce the same output is by entering something related to the common input. Note, however, that the game rules do not directly tell the players to enter a correct output for the given

Figure 3: In this inversion-problem game, given an input, Player 1 produces an output, and Player 2 guesses the input.

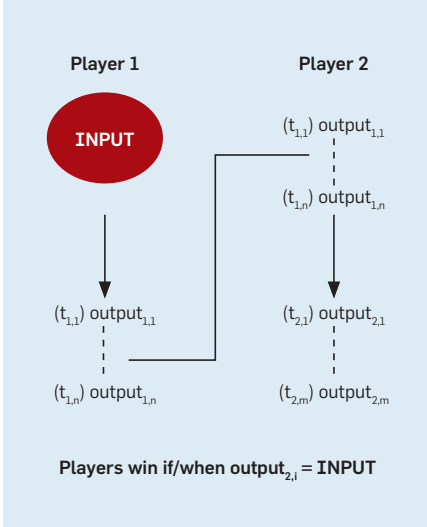
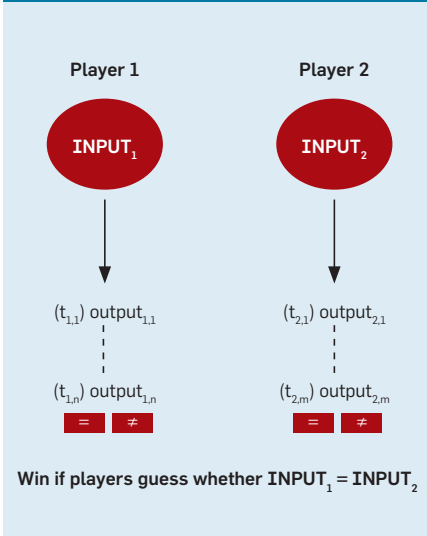


Figure 4: In this input-agreement game, players must determine whether they have been given the same input.



input; all they know is that they must “think like each other” and enter the same output.

This game structure accomplishes several goals at once: a good “winning” strategy for the players is to produce outputs related to the only thing they have in common—the input; when the two players provide the same output, this partially verifies that the output is correct, since it comes from two largely independent sources; and trying to agree on the same output with a partner is an enjoyable social experience.

Inversion-problem games. Resulting from any of three seemingly different games—Peekaboom,²⁵ Phetch,²³

and Verbosity²⁴—they, in their most general form (see Figure 3), can be described through the following rules:

Initial setup. Two strangers are randomly chosen by the game itself from among all potential players;

Rules. In each round, one player is assigned to be the “describer,” and the other player is assigned to be the “guesser.” The describer is given an input. Based on this input, the describer produces outputs that are sent to the guesser. The outputs from the describer should help the guesser produce the original input; and

Winning condition. The guesser produces the input that was originally given to the describer.

Verbosity (see the sidebar) is an inversion-problem game where the input is a word and the outputs are commonsense facts related to that word. Following the input word “milk,” the game might output such facts as “it is white” and “people usually eat cereal with it.”

Although the design of Verbosity involves other game elements not described here, the basic idea is that players need not be asked directly for facts about “milk.” The game is designed such that facts are collected as a side effect of playing. Players told to “please enter facts about milk” might not be motivated to do so or enter incorrect information.

In inversion-problem games, partners are successful only when the describer provides enough outputs for the guesser to guess the original input. If the outputs are incorrect or incomplete, the guesser will not be able to produce the original input. Therefore, the game structure encourages players to enter correct information. At the same time, having one player guess the input while the other describes it is an enjoyable social interaction, similar to the popular children’s game “20 Questions.”

Additional elements can be added to inversion-problem games to increase player enjoyment, including transparency and alternation:

Transparency. In post-game questionnaires, players of inversion-problem games have expressed a strong desire to see their partner’s guesses. We therefore experimented with adding a level of transparency between players

so the actions of one would be visible to the other. In games like Verbosity and Peekaboom this transparency is achieved by displaying partner guesses to the describers and allowing them to indicate whether each guess is “hot” or “cold.” This design feature increases the social connection between the players without compromising output correctness.

Alternation. Unlike output-agreement games (where both players continually perform the same task), inversion-problem games are asymmetric in that each player in the pair performs a different task. In some games of this type, one of the two roles involves more interaction or is faster-paced and thus more enjoyable than the other role. In such cases, to balance the game and maintain an equal level of player engagement, player roles can switch after each round; the guesser becomes the describer, and the describer becomes the guesser.

Input-agreement games. Representing a generalization of games like Edith Law’s TagATune⁹ (see Figure 4), they can be described through the following rules:

Initial setup. Two strangers are randomly chosen by the game itself from among all potential players;

Rules. In each round, both players are given inputs that are known by the game (but not by the players) to be the same or different. The players are instructed to produce outputs describing their input, so their partners are able to assess whether their inputs are the same or different. Players see only each other’s outputs; and

Winning condition. Both players correctly determine whether they have been given the same or different inputs.

In TagATune, the input is a sound clip, and the output is a series of labels or tags for the clip. The two players achieve the winning condition (and obtain points) only if they both correctly determine whether they have the same input song. Because players want to achieve the winning condition, they each want their partner to be able to determine if their inputs are the same. This means it is in their own best interest to enter accurate outputs that appropriately describe their individual inputs.

To discourage players from randomly guessing whether their inputs are the same, scoring in input-agreement games strongly penalizes incorrect guesses. One way to do this (while maintaining a positive scoring system) is to give an increasing number of points for streaks of correct answers and zero points for incorrect answers.


Increase Player Enjoyment

Perhaps the most important aspect of GWAP is that the output is produced in a way that's designed to be enjoyable. As noted with respect to the ESP Game, players are not directly instructed to enter keywords for a given image. Rather, they are told to type what they think their partner is typing. The fact that people enjoy the game makes them want to continue playing, in turn producing more useful output.


It is important to note that the three basic templates defined earlier describe the basic structure of a GWAP; additional game mechanisms must be added to them to increase player enjoyment. For example, much of the previous work describing game-design principles cites challenge as a key aspect of any successful game.^{11,12,14,20} Challenge translates into game features (outlined by Malone^{11,12}) like timed response, score keeping, player skill level, high-score lists, and randomness:

Timed response. Setting time limits for game sessions introduces challenge into a game in the form of timed response.^{11,12} Players are told to complete a designated number of problem instances within an assigned time limit. If they accomplish it, they may be given extra points for their performance. Timed response is effective for introducing challenge because it establishes an explicit goal that is not trivial for players to achieve if the game is calibrated properly.^{11,12} We know from the literature on motivation in psychology and organizational behavior that goals that are both well-specified and challenging lead to higher levels of effort and task performance than goals that are too easy or vague.¹⁰ It is essential that the number of tasks for players to complete within a given time period is calibrated to introduce challenge and that the time limit and time remaining are displayed throughout the game.

Score keeping. One of the most di-



It is essential that the number of tasks for players to complete within a given time period is calibrated to introduce challenge and that the time limit and time remaining are displayed throughout the game.



rect methods for motivating players is by assigning points for each instance of successful output produced during the game. For the ESP Game,²² pairs of players are given points for each image for which they successfully agree on a word (which then becomes a label for the image). Using points increases motivation by providing a clear connection among effort in the game, performance (achieving the winning condition), and outcomes (points).^{11,12} A score summary following each game also provides players with performance feedback,¹⁰ facilitating progress assessment on score-related goals (such as beating a previous game score and completing all task instances within the set time limit).

Player skill levels. Player skill levels, or “ranks,” are another way for game developers to incorporate goal-based motivation into GWAP design. For example, the ESP Game and Peekaboom each have five skill levels players are able to achieve based on the number of points they accumulate. Each newcomer to the game initially has no points and is assigned to the lowest level (“newbie”) then has to earn a certain number of points to advance to the next level.

Following each game session, players are shown their current skill level and the number of points needed to reach the next level.¹⁰ Data from the ESP Game indicates that presentation of this skill-level information strongly influences player motivation and behavior. Of the 200,000+ players as of July 2008 with an account on the ESP Game, 42% have scores that fall within 5,000 points of the rank cutoffs. Given that these skill-level point intervals cover less than 2% of the space of possible cumulative scores, the data suggests that many players continue playing just to reach a new rank.

High-score lists. Another method for motivating GWAP play is the use of high-score lists showing the login names and score of the subset of players with the highest number of points over a certain period of time. The score needed by players to be listed on a high-score list varies in terms of difficulty relative to the list's time period, ranging from highest scores achieved in the past game session over the past hour or week all the way to the history

of the game. For example, an hourly high-score list gives players a specific point total to aim for to get onto the list, as well as relatively quick feedback (within the hour) about their progress toward it. A daily high-score list and all-time high-score list define goals of increasing difficulty. These multi-level goals, varying in difficulty, provide strong, positive motivation for extended game play—and related data generation.

Randomness. GWAPs should also incorporate randomness. For example, inputs for a particular game session are typically selected at random from the set of all possible inputs, and players are randomly paired to prevent cheating.


Because inputs are randomly selected, their difficulty varies, thus keeping the game interesting and engaging for expert and novice players alike.^{11,12} It also means that every game session involves uncertainty about whether all inputs will be completed within the time limit, adding to the challenge experienced by players.^{11,12}

Random partner assignment also ensures the uniqueness of each game session. Anecdotal evidence from the ESP Game²² suggests that during each game session players develop a sense of their partners' relative skill, a perception that affects their joint performance. The feeling of connection that players can get from these games is one of the factors that motivates repeated play.^{18,20}


Output Accuracy

Additional mechanisms must be added to GWAPs beyond the basic template structure to ensure output correctness and counter player collusion. For example, players of the ESP Game might try to circumvent the game's built-in verification mechanism by agreeing prior to the game that for every image they will always type the letter "a"; in this case, they would always match each other, and incorrect data would therefore be entered into the system. We describe generally applicable mechanisms in the following sections that have proved successful in guarding against player collusion and guaranteeing the correctness of the computation across all game templates.

Random matching. GWAPs are



The real measure of utility for a GWAP is therefore a combination of throughput and enjoyability.



meant to be played by hundreds, if not thousands, of people at once, most in distributed locations. Players paired or grouped randomly have no way of knowing their partner's identity so have no easy way to agree ahead of time on any cheating strategy. Thus, under random matching, the probability of two or more cheaters using the same strategy being paired together should be low.

Player testing. Games may randomly present players inputs for which all possible correct outputs are already known. For them, if the output produced by a particular player does not match the known correct outputs, the players should be considered suspicious, and none of their results should be trusted. Depending on the number of "test" inputs presented to players, this strategy can guarantee with high probability that the output is correct. To illustrate, assume half of the inputs given to a player are test inputs. The probability is thus that a new output by the player is correct, given of course that the player is correct on all the test inputs at least 50% of the time, a probability that can be increased through repetition.

Repetition. A game should be designed so it does not consider an output correct until a certain number of players have entered it. This strategy for determining correctness enables any GWAP to guarantee correct output with arbitrarily high probability. As an example, consider an output-agreement game; if for a given input the game accepts an output as correct only after n pairs have entered it, and the game itself knows that each of these n pairs entered a correct output with at least 50% probability (as a result of player testing), then the output is correct with probability of at least $(1-\frac{1}{2})^n$.

Taboo outputs. For problems in which many different outputs can be associated with one input (such as labeling images with words), ensuring sufficient coverage of the output space is an important consideration. The use of "taboo," or off-limits, outputs provides some guarantee that a larger proportion of all possible outputs will be entered by all players. Taboo outputs are known correct outputs displayed onscreen during game sessions that players are not allowed to enter. They can be taken from correct outputs gen-

erated in previous rounds of the game itself. It is important for the game’s designer to randomize which taboo outputs are presented in order to account for potential output-priming effects (in which the particular taboo outputs shown to the players influence the guesses they enter) and ensure wide coverage of all potential outputs for a given input.

Other Design Guidelines

The general schemes we’ve presented here for designing GWAPs rely on the participation of two players per game session. Now we show that the games can be modified to accommodate single or more than two players.

Prerecorded games. Paired game play makes GWAPs social, meaning that players are able to validate each other’s computation. However, two-player games present logistical challenges. For instance, there may be times when an odd number of people want to play a particular game, meaning at least one of them cannot play. In addition, when a game is just beginning to gain popularity, it is difficult for game adminis-

trators to guarantee that many people will be able to play at the same time. We thus recommend that game developers apply a technique—prerecorded game play—introduced by the ESP Game.²² A dyadic game, normally played by multiples of two players, can be transformed into a single-player game by pairing a single player with a prerecorded set of actions.

In the case of an input-agreement game or output-agreement game (such as the ESP Game), implementing automated players is relatively easy. When two people are playing, the game should simply record every action they make, along with the relative timing of each action. Then, when a single player wishes to play, the system can pair that single player with a prerecorded set of moves.

In inversion-problem games, implementing prerecorded game play is more complex because one of the players (the guesser) must dynamically respond to the other (human) player’s actions. Peekaboom, Phetch, and Verbosity have each implemented a single-player version using techniques cus-

tomized to each game.^{23–25}

More than two players. The three GWAP templates can be extended to include more than two players; for example, output-agreement games can be extended to incorporate more players by modifying the winning condition such that the first two players who agree on the output are the winners of the round (and granted a higher number of points than the nonwinners). Similarly, the template for inversion-problem games can be extended to incorporate multiple players by substituting an individual guesser with an arbitrary number of players in the role of guesser, all racing to be first to correctly guess the input (winning condition).

These extensions change the nature of the games considerably. Whereas the two-player versions of each template are cooperative in nature (players work together to obtain points), the multiplayer versions are competitive. Cooperative, as well as competitive, games involve advantages and disadvantages. For certain players, competitive games may be more enjoyable than

A Sampling of GWAPs

The ESP Game and Verbosity

Two of the most popular GWAPs—ESP and Verbosity—can be played online at www.gwap.com.

The ESP Game has generated millions of labels for random images located throughout the Web. In it, two players are randomly paired for two-and-a-half minutes as they are shown a series of images to label. The game does not directly ask them to label the images. Rather, both players must try to enter the same word as their partner for each image on the screen; neither player can see the partner’s words. When both players agree on a word, each is given a new image. The goal is to agree on words for as many images as possible. The words the players agree on for each image are extremely accurate labels that can be used to improve image search throughout the Web. To increase the quality of these labels, as well as to motivate player engagement, the game

forbids the use of “taboo words” from being entered. In the screenshot (see Figure a), players cannot use the words “dog” or “puppy” when trying to agree on a word with their partner.

Verbosity is a word-guessing game in which two players alternate roles. The describer is given a secret word the guesser

must figure out as quickly as possible. The describer helps the guesser by providing clues about the secret word using sentence templates that must be completed without using the secret word itself. In the example here (see Figure b), the secret word is “sock,” and the sentence template “It is a kind of _____”

has been instantiated to the clue “It is a kind of clothing.” The describer sees all of the guesser’s inputs and indicates which ones are “hot” and which are “cold.” The computational purpose of the game is to collect a database of commonsense facts about the secret words (such as “Sock is a kind of clothing”).



Figure a: Players of the ESP Game try to guess what their partner is typing on each image.



Figure b: Players of Verbosity enter commonsense facts to help their partner guess a secret word.

their cooperative counterparts. On the other hand, having more players work on the same input is wasteful in terms of “computational efficiency,” an important criterion for evaluating the utility of a given game.

GWAP Evaluation

How might a game’s performance be judged successful? Given that two different GWAPs solve the same problem, which is best? We describe a set of metrics for determining GWAP success, including throughput, lifetime play, and expected contribution.

Game efficiency and expected contribution. If we treat games as if they were algorithms, efficiency would be a natural metric of evaluation. There are many possible algorithms for any given problem, some more efficient than others. Similarly, many possible GWAPs are available for any given problem. In order to choose the best solution to a problem we need a way to compare the alternatives in terms of efficiency. Efficiency of standard algorithms is measured by counting atomic steps. For instance, QuickSort is said to run in $O(n \log n)$ time, meaning it sorts a list of n elements in roughly $n \log n$ computational steps. In the case of GWAPs, the notion of what constitutes a computational step is less clear. Therefore, we must be able to define efficiency through other means.

First, we define the throughput of a GWAP as the average number of problem instances solved, or input-output mappings performed, per human-hour. For example, the throughput of the ESP Game is roughly 233 labels per human-hour.²² This is calculated by examining how many individual inputs, or images, are matched with outputs, or labels, over a certain period of time.

Learning curves and variations in player skill must be considered in calculating throughput. Most games involve a certain type of learning, meaning that with repeated game sessions over time, players become more skilled at the game. For the game templates we described earlier, such learning can result in faster game play over time. To account for variance in player skill and changes in player speed over time as a result of learning, we define throughput as the average number of problem instances solved per human-hour. This

average is taken over all game sessions through a reasonably lengthy period of time and over all players of the game.

Games with higher throughput should be preferred over those with lower throughput. But throughput is not the end of the story. Because a GWAP is a game, “fun” must also be included. It does not matter how many problem instances are addressed by a given game if nobody wants to play. The real measure of utility for a GWAP is therefore a combination of throughput and enjoyability.

Enjoyability is difficult to quantify and depends on the precise implementation and design of each game. Even seemingly trivial modifications to a game’s user interface or scoring system can significantly affect how enjoyable it is to play. Our approach to quantifying this elusive measure is to calculate and use as a proxy the “average lifetime play” (ALP) for a game. ALP is the overall amount of time the game is played by each player averaged across all people who have played it. For instance, on average, each player of the ESP Game plays for a total of 91 minutes.

“Expected contribution” is our summary measure of GWAP quality. Once a game developer knows on average how many problems are solved per human-hour spent in the game (throughput) and how much time each player can be expected to spend in a game (ALP), these metrics can be combined to assess each player’s expected contribution. Expected contribution indicates the average number of problem instances a single human player can be expected to solve by playing a particular game. Developers can then use this measure as a general way of evaluating GWAPs. We define the three GWAP metrics this way:

Throughput = average number of problem instances solved per human-hour;

ALP = average (across all people who play the game) overall amount of time the game will be played by an individual player; and

Expected contribution = throughput multiplied by ALP.

Although this approach does not capture certain aspects of games (such as “popularity” and contagion, or word of mouth), it is a fairly stable measure of a game’s usefulness. Previous work

in the usability tradition on measuring fun and game enjoyment has suggested the usefulness of self-report questionnaire measures.^{7,14} However, a behavioral measure (such as throughput) provides a more accurate direct assessment of how much people play the game and, in turn, how useful the game is for computational purposes.

Finally, a GWAP’s developers must verify that the game’s design is indeed correct; that is, that the output of the game maps properly to the particular inputs that were fed into it. One way to do this (as with the ESP Game, Peekaboom, Phetch, and Verbosity) is to analyze the output with the help of human volunteers. We have employed two techniques for this kind of output verification: comparing the output produced in the game to outputs generated by paid participants (rather than game players)²² and having independent “raters” evaluate the quality of the output produced in the game.²² Output from a GWAP should be of comparable quality to output produced by paid subjects.

Conclusion

The set of guidelines we have articulated for building GWAPs represents the first general method for seamlessly integrating computation and gameplay, though much work remains to be done. Indeed, we hope researchers will improve on the methods and metrics we’ve described here.

Other GWAP templates likely exist beyond the three we have presented, and we hope future work will identify them. We also hope to better understand problem-template fit, that is, whether certain templates are better suited for some types of computational problems than others.

The game templates we have developed thus far have focused on similarity as a way to ensure output correctness; players are rewarded for thinking like other players. This approach may not be optimal for certain types of problems; in particular, for tasks that require creativity, diverse viewpoints and perspectives are optimal for generating the broadest set of outputs.¹⁷ Developing new templates for such tasks could be an interesting area to explore.

We would also like to understand what kinds of problems, if any, fall outside the GWAP approach. The games

we have designed so far have focused on problems that are easily divided into subtasks. The “bite-size” nature of these games adds to their popularity and appeal to casual gamers in particular, since such players typically go for games they can play “just one more time” without having to make too much of a time commitment.

The GWAP approach represents a promising opportunity for everyone to contribute to the progress of AI. By leveraging the human time spent playing games online, GWAP game developers are able to capture large sets of training data that express uniquely human perceptual capabilities. This data can contribute to the goal of developing computer programs and automated systems with advanced perceptual or intelligence skills.

Acknowledgment

We would like to thank Manuel and Lenore Blum, Mike Crawford, Shiry Ginosar, Severin Hacker, Susan Hrishenko, Mihir Kedia, Edith Law, Bryant Lee, and Roy Liu for their help in this research.

References

1. Carroll, J.M. and Thomas, J.M. Fun. *ACM SIGCHI Bulletin* 19, 3 (Jan. 1988), 21–24.
2. Chao, D. Computer games as interfaces. *Interactions* 11, 5 (Sept.–Oct. 2004), 71–72.
3. Chao, D. Doom as an interface for process management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Mar. 31–Apr. 5). ACM Press, New York, 2001, 152–157.
4. Fails, J.A. and Olsen, D.R. Interactive machine learning. In *Proceedings of the Eighth International Conference on Intelligent User Interfaces* (Miami, Jan. 12–15). ACM Press, New York, 2003, 39–45.
5. Federoff, M. *Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games*. Unpublished thesis, Indiana University, Bloomington; www.melissafederoff.com/thesis.html.
6. Giles, J. Internet encyclopaedias go head to head. *Nature* 438 (Dec. 15, 2005), 900–901.
7. Hassenzahl, M., Beu, A., and Burmeister, M. Engineering joy. *IEEE Software* 18, 1 (Jan.–Feb. 2001), 70–76.
8. Laurel, B.K. Interface as mimesis. In *User-Centered System Design: New Perspectives on Human-Computer Interaction*, D.A. Norman and S.W. Draper, Eds. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 1986, 67–85.
9. Law, E.L.M., von Ahn, L., Dannenberg, R.B., and Crawford, M. TagATune: A game for music and sound annotation. In *Proceedings of the Eighth International Conference on Music Information Retrieval* (Vienna, Austria, Sept. 23–30). Austrian Computer Society, Vienna, Austria, 2007, 361–364.
10. Locke, E.A. and Latham, G.P. *A Theory of Goal Setting and Task Performance*. Prentice Hall, Englewood Cliffs, NJ, 1990.
11. Malone, T.M. Heuristics for designing enjoyable user interfaces: Lessons from computer games. In *Proceedings of the Conference on Human Factors in Computing Systems* (Gaithersburg, MD, Mar. 15–17). ACM Press, New York, 1982, 63–68.
12. Malone, T.M. What makes things fun to learn? Heuristics for designing instructional computer games. In *Proceedings of the Third ACM SIGSMALL*

Symposium and the First SIGPC Symposium on Small Systems (Palo Alto, CA, Sept. 18–19). ACM Press, New York, 1980, 162–169.

13. Mokka, S., Väättä, A., Heinilä, J., and Väikkynen, P. Fitness computer game with a bodily user interface. In *Proceedings of the Second International Conference on Entertainment Computing* (Pittsburgh, PA, May 8–10). Carnegie Mellon University, Pittsburgh, PA, 2003, 1–3.
14. Pagulayan, R., Keeker, K., Wixon, D., Romero, R., and Fuller, T. User-centered design in games. In *The Human-Computer Interaction Handbook: Fundamentals, Evolving Techniques and Emerging Applications*, J.A. Jacko and A. Sears, Eds. Lawrence Erlbaum Associates, Mahwah, NJ, 2003, 883–905.
15. Richards, C. Teach the world to twitch: An interview with Marc Prensky, CEO and founder Games2train.com. *Futurelab* (Dec. 2003); www.futurelab.org.uk/resources/publications_reports_articles/web_articles/Web_Article578.
16. Shneiderman, B. Designing for fun: How can we design user interfaces to be more fun? *Interactions* 11, 5 (Sept.–Oct. 2004), 48–50.
17. Steiner, I. *Group Process and Productivity*. Academic Press, New York, 1972.
18. Stork, D.G. and Lam C.P. Open mind animals: Ensuring the quality of data openly contributed over the World Wide Web. In *Learning from Imbalanced Data Sets: Papers from the AAAI Workshop (Technical Report WS-00-05)*. (Austin, TX, July 30–Aug. 1). American Association for Artificial Intelligence, Menlo Park, CA, 2000, 4–9.
19. Stork, D.G. The Open Mind Initiative. *IEEE Intelligent Systems & Their Applications* 14, 3 (May–June 1999), 19–20.
20. Sweetser, P. and Wyeth, P. GameFlow: A model for evaluating player enjoyment in games. *ACM Computers in Entertainment* 3, 3 (July 2005), 3.
21. Tsang, M., Fitzmaurice, G., Kurtenbach, G., and Khan, A. Game-like navigation and responsiveness in non-game applications. *Commun. ACM* 46, 7 (July 2003), 57–61.
22. von Ahn, L. and Dabbish, L. Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria, Apr. 24–2). ACM Press, New York, 2004, 319–326.
23. von Ahn, L., Ginosar, S., Kedia, M., and Blum, M. Improving image search with Phetch. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* (Honolulu, Apr. 15–20). IEEE Press, New York, 2007, IV-1209–IV-1212.
24. von Ahn, L., Kedia, M., and Blum, M. Verbosity: A game for collecting common-sense knowledge. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montreal, Apr. 22–27). ACM Press, 2007, 75–78.
25. von Ahn, L., Liu, R., and Blum, M. Peekaboomb: A Game for locating objects in images. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montreal, Apr. 22–27). ACM Press, New York, 2006, 55–64.
26. Webster, J. Making computer tasks at work more playful: Implications for systems analysts and designers. In *Proceedings of the SIGCPR Conference on Management of Information Systems Personnel* (College Park, MD, Apr. 7–8). ACM Press, New York, 1988, 78–87.

This work was partially supported by National Science Foundation grants CCR-0122581 and CCR-0085982 (ALADDIN) and generous gifts from Google, Inc., and the Heinz Endowment. Luis von Ahn was partially supported by a Microsoft Research Graduate Fellowship, a Microsoft Research New Faculty Fellowship, and a MacArthur Fellowship.

Luis von Ahn (lav@andrew.cmu.edu) is an assistant professor in the Computer Science Department at Carnegie Mellon University, Pittsburgh, PA.

Laura Dabbish (dabbish@andrew.cmu.edu) is an assistant professor of information technology and organizations in the Heinz School, with a joint appointment in the Human-Computer Interaction Institute in the School of Computer Science at Carnegie Mellon University, Pittsburgh, PA.

© 2008 ACM 0001-0782/08/0800 \$5.00



DOI:10.1145/1378704.1378720

Why Wikipedia's remarkable growth is sustainable.

BY DIOMIDIS SPINELLIS AND PANAGIOTIS LOURIDAS

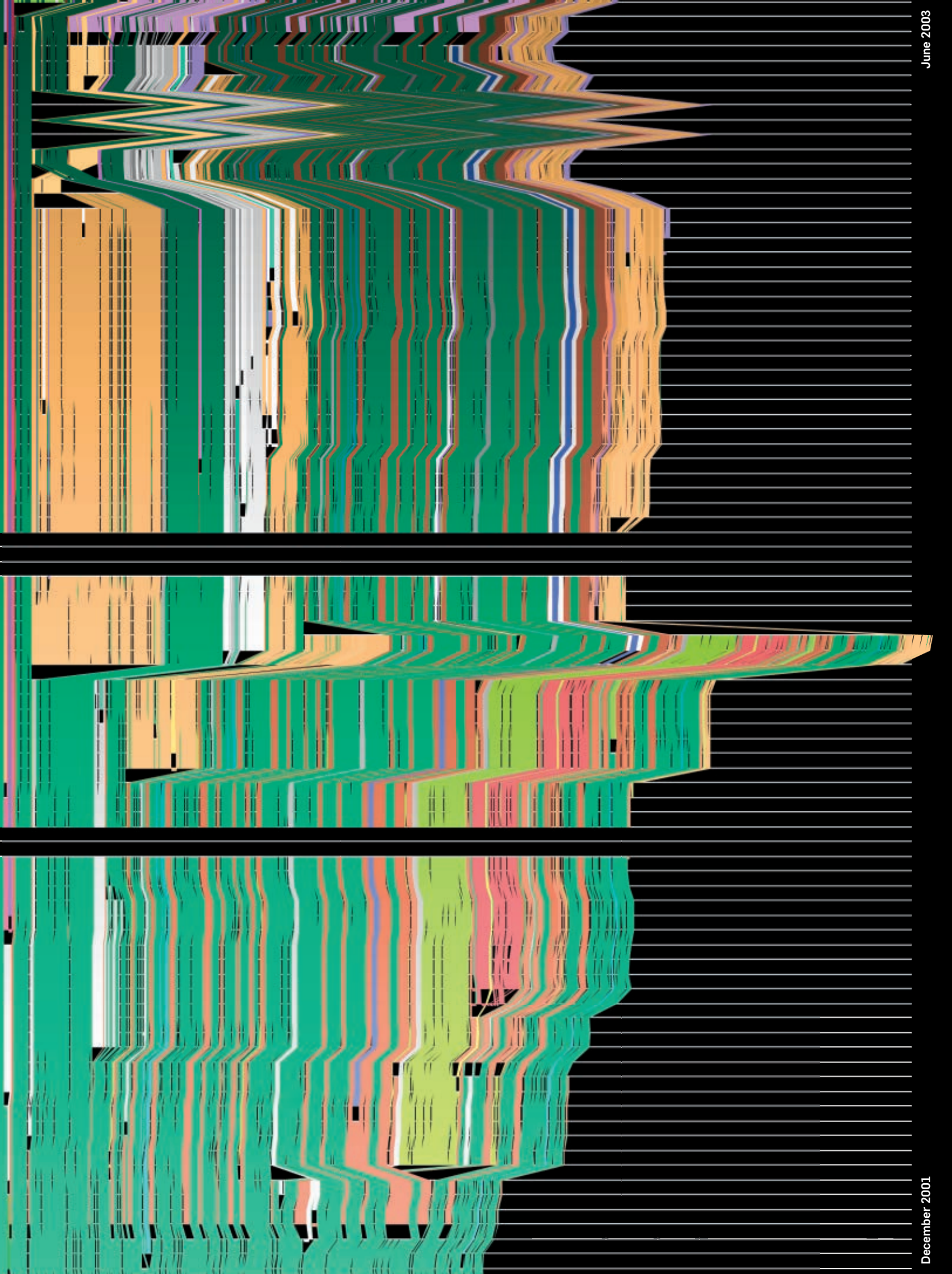
The Collaborative Organization of Knowledge

WIKIPEDIA (WWW.WIKIPEDIA.ORG) is a freely available online encyclopedia anyone can edit, contributing changes, as well as articles.¹⁰ With more than a million entries, hundreds of thousands of contributors, and tens of millions of fully recorded article revisions, Wikipedia's freely available database has also made it possible to study how human knowledge is recorded and organized through an open collaborative process. Although citation analysis⁶ can establish how new research builds on existing publications, the fully recorded evolutionary development of Wikipedia's structure has allowed us to examine how existing articles foster development of new entries and links. Motivation for our longitudinal study of Wikipedia evolution followed from our observation that even though Wikipedia's scope is increasing, its coverage is apparently not deteriorating. To study the process of Wikipedia growth we downloaded the February 2006

snapshot of all recorded changes and examined how entries are created and linked. Inspecting the timestamps on individual entry definitions and references, we found that links to non-existent articles often precede creation of new articles. Also, tracking the evolution of article links allowed us to empirically validate Barabási's hypothesis on the formation of scale-free graphs through incremental growth and preferential attachment.¹ Our findings paint a picture of sustainable growth, suggesting that Wikipedia's development process delivers coverage of more and more subjects.

The phenomenal growth of Wikipedia is attributable to a mixture of technologies and a process of open participation. The key technology behind Wikipedia is that of a Wiki—online lightweight Web-based collaboration.⁴ Wikipedia content appears online as static HTML pages, though each such page includes an edit button anyone can use to modify its content; editing most articles requires no prior authorization or arrangement. The system maintains the complete edit history of each page and supports a “watchlist” mechanism that alerts registered users when a page they are interested in changes.

The page history and watchlist facilities promote low-overhead collaboration and identification of and response to instances of article vandalism. We found that 4% of article revisions were tagged in their descriptive comment as “reverts”—the typical response to vandalism. They occurred an average of 13 hours after their preceding change. Looking for articles with at least one revert comment, we found that 11% of Wikipedia's articles had been vandalized at least once. (The entry for George W. Bush had the most revisions and reverts: of its 28,000 revisions one-third were reverts and, conceivably, another third vandalism.) Articles prone to vandalism can be administratively locked against revisions, a step rarely taken; in our study only 0.13% of the articles (2,441 entries) were locked.




When edited, an entry's content doesn't use the Web's relatively complex, error-prone HTML syntax but rather a simplified text annotation scheme called wiki markup, or wikitext. Creating a link from one entry to another is as simple as enclosing the other entry's identifying name in double square brackets. Markup tags can also group together related articles into categories (such as "Nobel laureates in physics," "liberal democracies," and "bowed instruments"). One use of a category tag is to mark entries as stubs, indicating to readers and future contributors that a particular entry is incomplete and requires expansion. In the snapshot we studied, about 20% of the entries were marked as stubs. For a better idea of Wikipedia's process and technology, access an entry in your own specialty and contribute an improvement.


Existing research on Wikipedia employs descriptive, analytic, and empirical methodologies. A series of measurements has been published that identifies power laws in terms of number of distinct authors per article, articles edited per author, and ingoing, outgoing, and broken links.¹³ On the analysis front, notable work has used simulation models to demonstrate preferential attachment,³ visualization techniques to identify cooperation and conflict among authors,¹² social-activity theories to understand participation,² and small-worlds network analysis to locate genre-specific characteristics in linking.⁸ Finally, given the anarchic nature of Wikipedia development, it is not surprising that some studies have also critically examined the quality of Wikipedia's articles.^{7,11} The work we describe here focuses on the dynamics of Wikipedia growth, examining the relationship between existing and pending articles, the addition of new articles as a response to references to them, and the building of a scale-free network of articles and references.

Methods

The complete content of the Wikipedia database is available online in the form of compressed XML documents containing separate revisions of every entry, together with metadata (such as the revision's timestamp, contributor, and modification comment). We processed the February 2006 complete



We hypothesize that the addition of new Wikipedia articles is not a purely random process following the whims of its contributors but that references to nonexistent articles trigger the eventual creation of a corresponding article.



dump of the English-language Wikipedia, a 485GB XML document. (In June 2008, we looked to rerun the study with more recent data, but complete dumps were no longer available.) The text of each entry was internally represented through the wiki-specific annotation format; we used regular expressions and explicit state transitions in a flex-generated analyzer for parsing both the XML document structure and the annotated text. From the database's entries we skipped all entries residing in alternative namespaces (such as "talk" pages containing discussions about specific articles, user pages, and category pages). In total, we processed 28.2 million revisions on 1.9 million pages.

For each Wikipedia entry we maintained a record containing the contributor identifiers and timestamps for the entry's definition and for its first reference, the number of efferent (outgoing) article references (unique references to other Wikipedia articles in the current version of the entry), the number of unique contributors, the number of revisions, a vector containing the number of the entry's afferent (incoming) references from other Wikipedia articles for each month, and a corresponding vector of Boolean values identifying the months during which the entry was marked as a stub. (The source code for the tools we used and the raw results we obtained are at www.dmst.aueb.gr/dds/sw/wikipedia.)

Growth and Unresolved References

We were motivated to do this research when one of us (Spinellis), in the course of writing a new Wikipedia entry, observed that the article ended up containing numerous links to other nonexistent articles. This observation led us to the "inflationary hypothesis" of Wikipedia growth, that is, that the number of links to nonexistent articles increases at a rate greater than the rate new articles are entered into Wikipedia; therefore Wikipedia utility decreases over time as its coverage deteriorates by having more and more references to concepts that lack a corresponding article. An alternative—the "deflationary hypothesis"—involves links to nonexistent articles increasing at a rate less than the rate of the addition of new articles. Under this hypoth-

esis we are able to project a point in the future when the Wikipedia engine of growth (discussed in the next section) will stall.

It turns out that the reality of Wikipedia development is located comfortably between the two extremes of non-existent link inflation and deflation. Figure 1 outlines the ratio between incomplete and complete articles from 2001 to 2006. Incomplete articles either don't exist in Wikipedia or exist but are marked as stubs. Although many stub articles contain useful information (often a link to an authoritative page with more detail), some pages also require additional work to be helpful but are not marked as stubs. For the purposes of our study we assume that the two effects cancel each other out.

The covered ratio from 2003 to 2006 seems stable, with about 1.8 missing or stub articles for every complete Wikipedia article. During the same time the number of articles surged from 140,000 to 1.4 million entries, showing that the apparently chaotic Wikipedia development process delivers growth at a sustainable rate.

References Lead to Definitions

Wikipedia's topic coverage has been criticized as too reflective of and limited to the interests of its young, tech-savvy contributors, covering technology and current affairs disproportionately more than, say, world history or the arts.⁵ We hypothesize that the addition of new Wikipedia articles is not a purely random process following the whims of its contributors but that references to nonexistent articles trigger the eventual creation of a corresponding article. Although it is difficult to claim that this process guarantees even and unbiased coverage of topics (adding links is also a subjective process), such a mechanism could eventually force some kind of balance in Wikipedia coverage.

The empirical findings outlined in Figure 2 support our hypothesis concerning the drive behind the addition of new articles. In particular, a reference to a nonexistent entry appears to be positively correlated with the addition of an article for it. Figure 2a tallies the number of articles with a given time difference between an entry's first reference and its subsequent definition. Most articles by far seem to be

Figure 1: Coverage of Wikipedia articles.

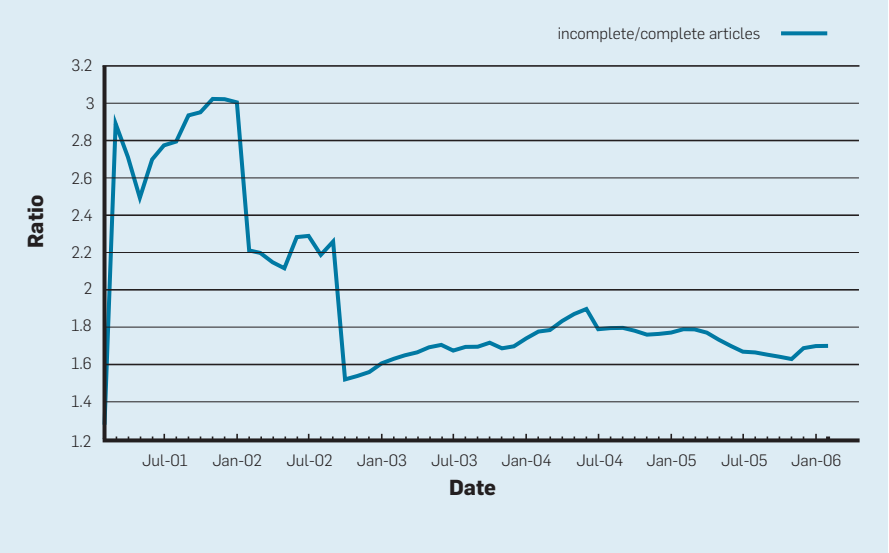


Figure 2a: References to an entry typically precede the entry's definition; number of entries with a given difference between the time of the first reference to the entry and the addition of its definition.

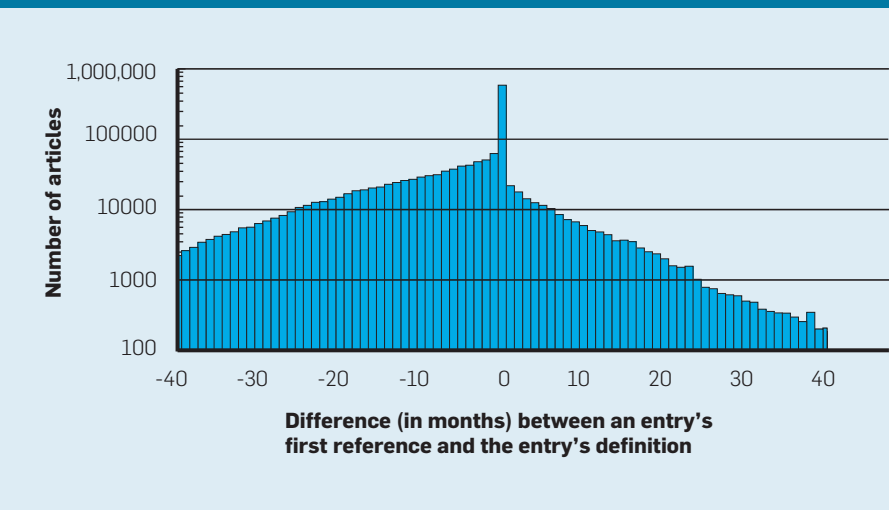
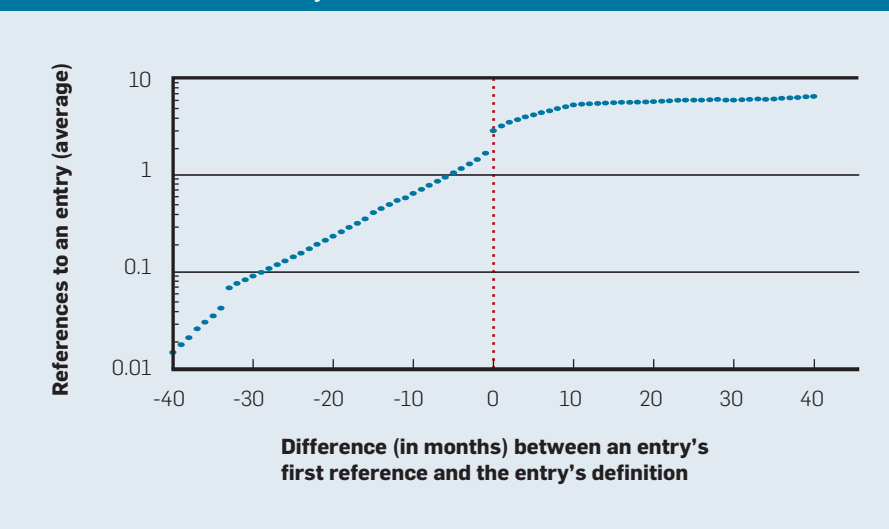


Figure 2b: References to an entry typically precede the entry's definition; number of references to an entry at the time of its definition.



created in the month of their first reference. Interestingly, the reference and subsequent definition of an article in Wikipedia appear to be a collaborative phenomenon; from the 1.7 million entries for which both the contributor entering the first reference and the contributor entering the first definition are known, that contributor is the same for only 47,000, or 3%, of entries.

Similarly, the mean number of first references to entries (see Figure 2b) rises exponentially until the referenced entry becomes an article. (For calculating the mean we offset each

entry's time of definition and time points in which it was referenced to center them at time 0.) The point in time when the referenced entry becomes an article marks an inflection point; from then on the number of references to a defined article rises only linearly (on average).

Building a Scale-Free Network

We established that entries are added to Wikipedia as a response to references to them, but what process adds references and entries? Several models have been proposed to explain the

appearance of scale-free networks like the one formed by Wikipedia's entries and references. The models can be divided into two groups:⁹ treating power laws as the result of an optimization process; and treating power laws as the result of a growth model, the most popular of which is Barabási's preferential attachment model.¹ In-vitro model simulations verify that the proposed growth models do indeed lead to scale-free graphs. Having the complete record of Wikipedia history allows us to examine in-vivo whether a particular model is indeed being followed.

Barabási's model of the formation of scale-free networks starts with a small number (m_0) of vertices. Every subsequent time step involves the addition of a new vertex, with $m \leq m_0$ edges linking it to m different vertices already in the system. The probability P that a new vertex will be connected to vertex i is $P(k_i) = k_i / \sum_j k_j$, where k_i is the vertex's connectivity at that step.

The situation in Wikipedia is more complex, as the number of vertices and edges added in a time step is not constant and new edges are added between existing vertices as well. We therefore consider a model where at each time step t a month, a variable number of entries and r_t references are added. The references are distributed among all entries following a probability $P(k_{i,t}) = k_{i,t} / \sum_j k_{j,t}$, with the sums and the connectivities calculated at the start of t . The expected number of references added to entry i at month t is then $\{k_{i,t}\} = r_t P(k_{i,t})$. We find a close match between the expected and the actual numbers in our data. Figure 3a is a quantile-quantile plot of the expected and the actual numbers at the 1,000-quantiles; Figure 3b outlines the frequency distributions of the number of articles (expected vs. actual) gaining a number of references in a month. The two data sets have a Pearson's product-moment correlation of 0.97, with the 95% confidence interval being (0.9667, 0.9740). If na_x is the number of articles that gained $x > 30$ (to focus on the tails) references in a month and na'_x is the expected number of such articles, we have $na_x \approx 1.11na'_x$ (p -value < 0.001).

It has never been possible to examine the emergence of scaling in other

Figure 3a: Expected and actual number of references added each month to an entry; quantile-quantile plot of the expected and actual number of references added each month to each article.

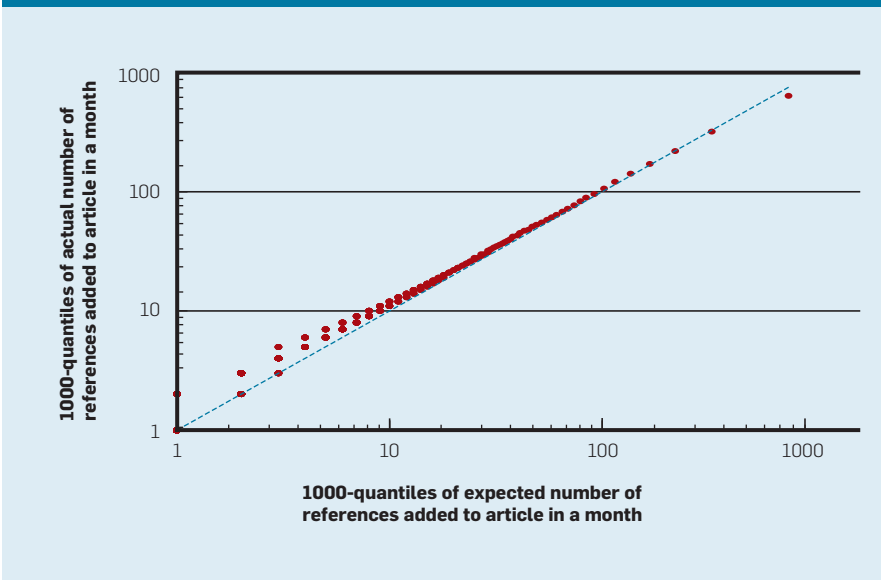
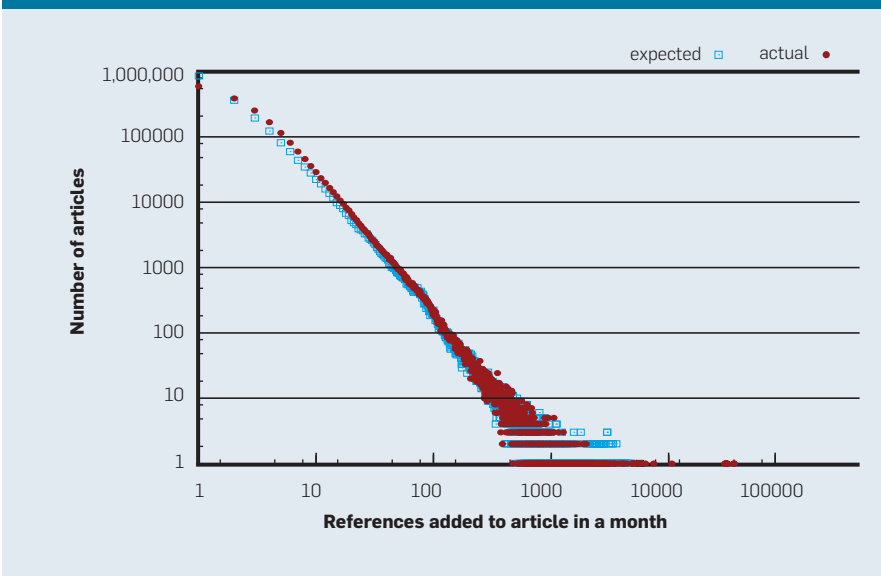


Figure 3b: Expected and actual number of references added each month to an entry; frequency distributions of the expected and actual number of references added each month to each article.



big real-world networks like the Web, as there is no full record of their evolution. Wikipedia now allows us to witness, and validate, preferential attachment at work on its graph.

Conclusion

The usefulness of an online encyclopedia depends on multiple factors, including breadth and depth of coverage, organization and retrieval interface, and trustworthiness of content. In Wikipedia more depth eventually translates into breadth, because the Wikipedia style guidelines recommend the splitting of overly long articles. The evolution of articles and links in Wikipedia allows us to model the system's growth. Our finding that the ratio of incomplete vs. complete articles remains constant yields a picture of sustainable coverage and growth. An increasing ratio would result in thinner coverage and diminishing utility and a decreasing ratio of incomplete vs. complete articles to eventual stagnation of Wikipedia growth.


The idea of growth triggered by undefined references is supported by our second finding—that most new articles are created shortly after a corresponding reference to them is entered into the system. We also found that new articles are typically written by different authors from the ones behind the references to them. Therefore, the scalability of the endeavor is limited not by the capacity of individual contributors but by the total size of the contributor pool.

Wikipedia's incremental-growth model, apart from providing an in-vivo validation of Barabási's scale-free network-development theory, suggests that the processes we have discovered may continue to shape Wikipedia in the future. Wikipedia growth could be limited by invisible subjective boundaries related to the interests of its contributors. Our growth model suggests how these boundaries might be bridged. Consider that references to nonexistent entries prompt creation of these entries and assume that all human knowledge forms a fully connected network. Wikipedia's coverage will broaden through a breadth-first graph traversal or flood-filling process, albeit over an uneven time progression.



It turns out that the reality of Wikipedia's development is located comfortably between the two extremes of nonexistent link inflation and deflation.



How far might the Wikipedia process carry us? In Jorge Luis Borges's 1946 short story "On Exactitude in Science," the wise men of the empire undertake to create a complete map of the empire; upon finishing, they realize the map was so big it coincided with the empire itself. 

References

1. Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *Science* 286, 5439 (Oct. 15 1999), 509–512.
2. Bryant, S., Forte, A., and Bruckman, A. Becoming Wikipedian: Transformation of participation in a collaborative online encyclopedia. In *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work* (Sanibel Island, FL, Nov. 6–9). ACM Press, New York, 2005, 1–10.
3. Capocci, A., Servidio, V., Colaiori, F., Buriol, L., Donato, D., Leonardi, S., and Caldarelli, G. Preferential attachment in the growth of social networks: The case of Wikipedia. *Physical Review E*, 74, 036116 (2006).
4. Cunningham, W. and Leuf, B. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley, Boston, MA, 2001.
5. Denning, P., Horning, J., Parnas, D., and Weinstein, L. Wikipedia risks. *Commun. ACM* 48, 12 (Dec. 2005), 152–152.
6. Garfield, E. *Citation Indexing: Its Theory and Application in Science, Technology, and Humanities*. John Wiley & Sons, Inc., New York, 1979.
7. Giles, J. Internet encyclopaedias go head to head. *Nature* 438, 7070 (Dec. 15, 2005), 900–901.
8. Mehler, A. Text linkage in the wiki medium: A comparative study. In *Proceedings of the EACL 2006 Workshop on New Text: Wikis and Blogs and Other Dynamic Text Sources* (Trento, Italy, Apr. 6, 2006), 1–8.
9. Mitzenmacher, M. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics* 1, 2 (2003), 226–251.
10. Remy, M. Wikipedia: The free encyclopedia. *Online Information Review* 26, 6 (2002), 434.
11. Stvilia, B., Twidale, M., Smith, L., and Gasser, L. Assessing information quality of a community-based encyclopedia. In *Proceedings of the International Conference on Information Quality* (Cambridge, MA, Nov. 4–6, 2005), 442–454.
12. Viégas, F., Wattenberg, M., and Dave, K. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria, Apr. 24–29). ACM Press, New York, 2004, 575–582.
13. Voß, J. Measuring Wikipedia. In *Proceedings of the 10th International Conference of the International Society for Scientometrics and Informetrics* (Stockholm, July 24–28, 2005), 221–231.

This work is partially funded by the European Commission's Sixth Framework Programme under contract IST-2005-033331 "Software Quality Observatory for Open Source Software (SQO-OSS)."

Diomidis Spinellis (dds@aub.gr) is an associate professor of information system technologies in the Department of Management Science and Technology at the Athens University of Economics and Business, Athens, Greece.

Panagiotis Louridas (louridas@acm.org) is a software engineer in the Greek Research and Technology Network and a researcher at the Department of Management Science and Technology, Athens University of Economics and Business, Athens, Greece.

$$\sqrt{1-\frac{v^2}{c^2}} \omega \left(t - \frac{v}{c} \cos \alpha_1 - \frac{z}{c} \cos \alpha_2 - \frac{z}{c} \cos \alpha_3 \right) =$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) = p(\chi|\phi, \Omega) p(\phi|\Omega) \frac{\partial f(\rho, t)}{\partial \rho} = -\partial \rho - 1 - \partial(\tau, \tau), \partial f$$

$$p(\phi|\tau, \beta) = p(\phi|\chi, \Omega) = p(\chi|\phi, \Omega) p(\phi|\Omega) \quad \mu > \psi_{\mu}^{\circ} +$$

$$\frac{5}{9} (F. 32) = \mu = \psi_{\mu}^{\circ} +$$

$$p(\tau, \beta) = \pi(\tau, \beta, \phi \xi) \pi(4.13) \cdot (9.2) \pi(6. \alpha) = 5 \lambda (\phi$$

$$p(\phi|\chi, \Omega) = (\partial v, \partial v) = \omega v v^{\circ} (V_0 \psi_0) \quad \mu >$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda. \quad \mu =$$

$$\frac{(F. 32) \frac{\partial \Sigma^2 \partial t}{\partial \Sigma^2} \sqrt{\frac{L}{2g}} \frac{1}{\sqrt{1-\frac{v^2}{c^2}}}}{\omega \left(t - \frac{v}{c} \cos \alpha_1 - \frac{z}{c} \cos \alpha_2 - \frac{z}{c} \cos \alpha_3 \right)} =$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

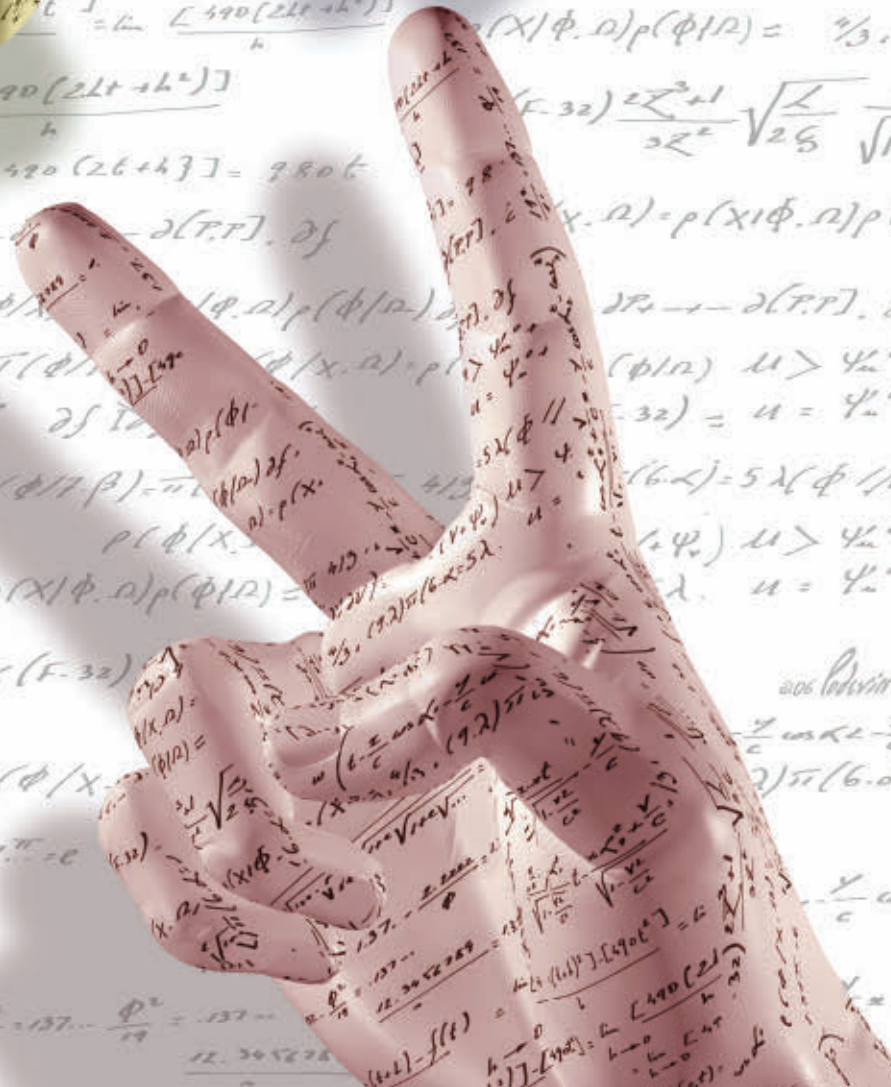
$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$

$$p(\chi|\phi, \Omega) p(\phi|\Omega) = \frac{1}{3}, (9.2) \pi(6. \alpha = 5 \lambda$$



DOI:10.1145/1378704.1378721

The most dramatic interaction between CS and GT may involve game-theory pragmatics.

BY YOAV SHOHAM

Computer Science and Game Theory

GAME THEORY HAS influenced many fields, including economics (its initial focus), political science, biology, and many others. In recent years, its presence in computer science has become impossible to ignore. GT is an integral part of artificial intelligence (AI), theory, e-commerce, networking, and other areas of computer science, and it is routinely featured in the field's leading journals and conferences. One reason is application pull: the Internet calls for analysis and design of systems that span multiple entities, each with its own information and interests. Game theory, for all its limitations, is by far the most developed theory of such interactions. Another reason is technology push: the mathematics and scientific mind-set of game theory are similar to those that characterize many computer scientists. Indeed, it is interesting to note that modern computer science and modern game theory originated in large measure at the same place and time—namely at Princeton

University, under the leadership of John von Neumann, in the 1950s.^a

In this article I try to do two things: identify the main areas of interaction between computer science and game theory so far; and point to where the most interesting interaction yet may lie—in an area that is still relatively underexplored.

The first part aims to be an unbiased survey, but it is impossible to avoid bias altogether. Ten researchers surveying the interactions between CS and GT would probably write 10 different types of reports. Indeed, several already have (as I will discuss). Moreover, in this brief discussion I cannot possibly do justice to all the work taking place in the area. So I try to compensate for these limitations in two ways: I provide a balanced set of initial pointers into the different subareas, without regard to the amount or nature of work that has taken place in each; and I point the reader to other relevant surveys of the CS-GT interaction, each having its own take on things.

The second part is decidedly subjective, but it is still meant to be broadly relevant both to computer scientists and game theorists interested in the interaction between the disciplines.

Lessons from Kalai (1995)

My departure point is a 13-year-old survey paper by E. Kalai,¹⁶ a game theorist with algorithmic sensibilities. Geared primarily toward computer scientists, the paper took stock of the interactions between game theory, operations research, and computer science at the time. It points to the following areas:

1. Graphs in games
2. The complexity of solving a game
3. Multiperson operations research
4. The complexity of playing a game
5. Modeling bounded rationality.

The reason I start with this paper, besides providing the interesting perspective of a non-computer scientist, is the comparison with current CS-GT interac-

^a I thank Moshe Tennenholtz for this observation, which is especially true of GT and AI.

tion, as both the matches and mismatches are instructive. Looking at the interactions between CS and GT taking place today, one can identify the following foci:

- a. Compact game representations;
- b. Complexity of, and algorithms for, computing solution concepts;
- c. Algorithmic aspects of mechanism design;
- d. Game-theoretic analysis inspired by specific applications;
- e. Multiagent learning;
- f. Logics of knowledge and belief, and other logical aspects of games.^b

The crude mapping between this list and Kalai's is as follows:

1995		2008
1	◀▶	<i>a</i>
2	◀▶	<i>b</i>
3	◀▶	<i>c, d</i>
4		
5		<i>e,</i> <i>f</i>

Here, I discuss the areas that match up (1◀▶*a*, 2◀▶*b*, 3◀▶*c, d*), then turn to the currently active areas that were not discussed by Kalai (*e, f*), and finish with the orphans on the other side (4, 5) that were discussed by Kalai but not yet vigorously pursued.

There has been substantial work on compact and otherwise specialized game representations. Some of them are indeed graph-based—graphical games,¹⁸ local-effect games,²¹ MAIDS,¹⁹ and Game networks,²⁰ for example. The graph-based representations extend also to coalition game theory.⁷ But specialized representations exist that are not graph based, such as those that are multi-attribute based⁵ and logic based.¹⁵ I believe this area is ripe for additional work—regarding, for example, the strategy space of agents described using constructs of programming languages.

The complexity of computing a sample Nash equilibrium (as well as other solution concepts) has been the focus of much interest in CS, especially within the theory community. A new complexity class—PPAD—was proposed to handle problems for which a solution

is known to exist,²⁷ the computation of a sample Nash equilibrium was shown to be complete for this class,² and the problem of computing Nash equilibria with specific properties was shown to be NP-hard.^{4, 10} At the same time, algorithms—some quite sophisticated, and all exponential in the worst case—have been proposed to compute Nash equilibria.^{11, 41} Somewhat surprisingly, recent experiments have shown that a relatively simple search algorithm significantly outperforms more sophisticated algorithms.³¹ This is an active area that promises many additional results.

The third match is somewhat less tight than the first two. There are at least two kinds of optimization one could speak about in a game-theoretic setting. The first is computing a best response to a fixed decision by the other agents; this is of course the quintessential single-agent optimization problem of operations research and AI, among other fields. The second is the optimization by the designer of a mechanism aimed at inducing games with desirable equilibria.

This so-called “mechanism design” has been the focus of much work in computer science. One reason is the interesting interaction between traditional CS problems (such as optimization and approximation) and traditional mechanism-design issues (such as incentive compatibility, individual rationality, and social-welfare maximization). A good example is the interaction between the Vickrey-Clarke-Groves mechanism and shortest-path computation;²⁶ another is the literature on combinatorial auctions,⁶ which combine a weighted-set-packing-like NP-hard optimization problem with incentive issues. The interplay between mechanism design and cryptography is worth particular mention. Though both are in the business of controlled dissemination of information, they are different in significant ways. For one thing, they are dual in the following sense: mechanism design attempts to *force* the *revelation* of information, while cryptography attempts to *allow* its *hiding*. For another, they traditionally embody quite different models of paranoia. Game theory assumes an even-keeled expected utility maximization on the part of all agents, while cryptography is more simple-minded: it assumes that “good” agents act as instructed, while “bad” agents are

maximally harmful. Recent work, however, has begun to bridge these gaps.

This third category blends into the fourth one, which is research motivated by specific applications that have emerged in the past decade. For example, the domain of networking has given rise to a literature on so-called “price of anarchy” (which captures the inefficiency of equilibria in that domain), games of routing, networking-formation games, and peer-to-peer networks. Other domains include sponsored search auctions, information markets, and reputation systems. This combination of the third and fourth categories is arguably the most active area today at the interface of CS and GT, and many aspects of it are covered in Nisan et al.,²⁵ which is an extensive edited collection of surveys. The popularity of this area is perhaps not surprising. The relevancies of specific applications speak for themselves (although arguments remain about whether the traditional game-theoretic analysis is an appropriate one). More generally, it is not surprising that mechanism design struck a chord in CS, given that much of CS's focus is on the design of algorithms and protocols. Mechanism design is the one area within GT that adopts such a design stance.

The fifth category active today is multiagent learning, also called “interactive learning” in the game-theory literature.^c Multiagent learning, long a major focus within game theory, has been rediscovered with something of a vengeance in computer science and in particular AI; witness special issues devoted to it in the *Journal of Artificial Intelligence*³⁹ and the *Machine Learning Journal*.¹² For computer science, the move from single-agent learning to multiagent learning is interesting not only because it calls for new solutions but also because the very questions change. When multiple agents learn concurrently, one cannot distinguish between learning and teaching, and the question of “optimal” learning is no longer well defined (just as the more general notion of an “optimal policy” ceases to be meaningful when one moves to the multiagent setting). For a discussion of this phenomenon, see the *Journal of Artificial Intelligence* special issue cited earlier.³⁹

b This current survey originated in a presentation made at a December 2007 festschrift in honor of E. Kalai.

c Kalai's omission of this area is ironic, as he co-authored one of its seminal papers.


The sixth and final major area of focus, also one not discussed in Kalai,¹⁶ is called “interactive epistemology” in game theory and simply “reasoning about knowledge and belief” in computer science. Starting in the mid-1980s, this area was for a while the most active focus of interaction between computer science (including distributed systems, AI, and theory) and game theory. Beside game theory, it established deep ties with philosophy and mathematical logic, culminating in the seminal book by Fagin et al.^{8, d} It is interesting to speculate why this area was omitted from Kalai’s list, even although it predates his paper by a decade, and why today it is not as broadly populated as the other areas. I think the reason is that the subject matter is more foundational, primarily non-algorithmic, and appeals to a smaller sliver of the two communities. Be that as it may, it remains a key area of interaction between the two fields.

These six areas are where most of the action has been in past years, but by listing only them and being brief about each one, I have by necessity glossed over some other important areas. The references compensate for this omission to some extent. In addition, the reader is referred to the following additional surveys, all by computer scientists who each have a slightly different slant. Most of these works go into considerably more detail about some of the topics.


► The earliest relevant survey is probably by Linial.²² Geared primarily toward game theorists, this 58-page report has deep coverage of game-theoretic aspects of distributed systems, fault-tolerant computing, and cryptography, and it also touches on computation of game theoretic concepts, games and logic, and other topics.

► Papadimitriou’s survey²⁹ geared to-

d That book focused on static aspects of knowledge and belief, which, notwithstanding the substantial computer-science credentials of the authors, raise an interesting contrast between the computer-science and game-theory literature in these areas. In game theory, static theories are indeed the primary focus, whereas in computer science—in particular, in database theory and artificial intelligence—belief revision and other dynamic theories³⁰ (including the entire mini-industry of nonmonotonic logics⁹) play an equal if not greater role. Indeed, recent work at the interface of logic and game theory³⁷ extends the static treatment of Fagin et al.⁸ in a dynamic direction.



I expect game-theory pragmatics to be as critical to reducing game theory to practice as language pragmatics have been to analyzing human discourse or understanding language by computers.



ward computer scientists, is a concise five-page paper summarizing the main complexity and algorithmic issues at the interface of CS and GT circa 2001.

► The 21-page paper by Halpern¹³ is similar to Linial in that it is geared toward game theorists and its main focus is distributed systems, but having been published a decade later it is more current. The work later evolved into a 17-page survey¹⁴ with an abbreviated discussion of distributed computing and additional material on complexity considerations, price of anarchy, mediators, and other topics.

► Roughgarden’s 30-page work is a detailed survey of a specific topic—namely, the complexity of computing a sample Nash equilibrium.³² Geared mostly toward economists, it includes ample background material on relevant concepts from complexity theory.

The material discussed so far is not only prominently featured in computer science journals and conferences but also is beginning to find its way into textbooks.³⁵ These areas will undoubtedly continue to flourish. But now I want to turn our attention to the two closely-related areas—4 and 5—listed by Kalai that have *not* been looked at as closely by the community at large, CS in particular. I do this for two reasons: I believe they are critical to the future success of game theory, and I believe that CS can play an important role in them. They both have to do with incorporating practical considerations into the model of rationality that is inherent to game theory. To repeat the caveat stated earlier: unlike the material so far, the remaining discussion is future-directed, speculative, and subjective.

Lessons from Linguistics

The field of linguistics distinguishes among syntax, semantics, and pragmatics. Syntax defines the form of language, semantics defines its meaning, and pragmatics defines its use. While the three interact in important ways, the distinctions have proved very useful. I believe that game theory may do well to make similar distinctions, and that CS can help in the process. Just as in the case in linguistics, it is unlikely that game-theory pragmatics will yield to unified clean theories, as do syntax and semantics. But I expect game-theory pragmatics to be as critical to reduc-


ing game theory to practice as language pragmatics have been to analyzing human discourse or understanding language by computers.

The distinction between the syntax and semantics of games is, I think, quite important, as some of the disputes within game theory regarding the primacy of different game representations (for example, the strategic and extensive forms) suffer from the lack of this distinction. It might, however, be presumptuous for CS to intrude on this debate, except insofar as it lends logical insights.³⁸ Indeed, perhaps this is more the role of mathematical logic than of CS per se.


But where CS can truly lead the way is in game theory's pragmatics. Game theory as we know it embodies radical idealizations, which include the infinite capacity of agents to reason and the infinite mutually recursive modeling of agents. Backing off from these strong assumptions has proven challenging. A fairly thin strand of work under the heading of "bounded rationality" involves games played by automata.³³ This is an important area of research that sometimes makes deep connections between the two fields. For example, early results showed that one of the well-known pesky facts in game theory—namely, that constant "defection" is the only subgame-perfect equilibrium in the finitely repeated prisoner's dilemma game—ceases to hold true if the players are finite automata with sufficiently few states.^{24,28} A more recent result shows that when players in a game are computer programs, one obtains phenomena akin to the Folk Theorem for repeated games.³⁶

This connection between theoretical models of computation and game theory is quite important and beautiful, but it constitutes a fairly narrow interpretation of the term "bounded rationality." The term should perhaps be reserved for describing a much broader research agenda—one that may encourage more radical departures from the traditional view in game theory. Let me mention two directions that I think would be profitable (and difficult) to pursue under this broader umbrella.

When one takes seriously the notion of agents' limited reasoning powers, it is not only some of the answers that begin to change; the questions themselves must be reconsidered. Consider



Science operates at many levels. For some, it is sufficient that scientific theories be clever, beautiful, and inspirational. Others require that any science eventually make contact with compelling applications and be subjected to empirical evaluation.



the basic workhorses of game theory—the Nash equilibrium and its many variants—that have so far served as the very basic analysis tool of strategic interactions. Questioning the role of equilibrium analysis will be viewed by some in GT as act of heresy, but real life suggests that we may have no choice. For example, in the trading agent competition, Nash equilibrium of the game did not play a role in almost any participating program,⁴² and this is certainly true as well of the more established chess and checkers competitions.

It is premature to write off the Nash equilibrium as irrelevant, however. For example, two programs competing in the TAC did in fact make use of what can be viewed as approximate empirical NE.⁴² Another striking example is the computation of equilibria in a simplified game tree by a top-scoring program in a poker competition.⁴³ It could be argued that maxmin strategies, which coincide with equilibrium strategies in zero-sum games, do play an important pragmatic role. But computation of either maxmin or equilibrium strategies in competitions has certainly been the exception to the rule. The more common experience is that one expends the vast majority of the effort on traditional AI problems such as designing a good heuristic function, searching, and planning. Only a little—albeit important—time is spent reasoning about the opponent.

The impact of such pragmatic considerations on game theory can be dramatic. Rather than start from very strong idealizing assumptions and awkwardly try to back off from them, it may prove more useful or accurate to start from assumptions of rather limited reasoning and mutual modeling, and then judiciously add what is appropriate for the situation being modeled. Which incremental-modeling approach will out has yet to be seen, but the payoff both for CS and GT can be substantial.

The second direction is radical in a different way. Game theory adopts a fairly terse vocabulary, inheriting it from decision theory and the foundations of statistics.^e In particular, agents

^e Parenthetically, it can be remarked that Savage's setting,³⁴ on which the modern Bayesian framework is based, does not have an obvious extension to the multi-agent case. However, this is not the focus of the point I am making here.

have “strategies,” which have minimal structure, and motivations, which are encapsulated in a simple real-valued utility function. (This in fact carries even less information than is suggested by the use of numbers, as the theory is unchanged by any positive affine transformation of the numbers.) In real life, and in computer programs attempting to behave intelligently, we find use for a much broader vocabulary. For example, agents are *able* to take certain actions and not others; have *desires*, *goals*, and *intentions* (the belief-desire-intention combination giving rise to the pun “beady-eye agent architecture”); and make *plans*. Apparently these abstract notions are useful both in effecting intelligent behavior and in reasoning about it. Philosophers have written about them (for example, Bratman¹) and there have been attempts—albeit preliminary ones—to formalize these intuitions (starting with Cohen and Levesque³). Some in AI have advocated embracing an even broader vocabulary of emotions (such as the recent provocative albeit informal book by Minsky.²³) Is game theory missing out by not considering these concepts?

Concluding Remarks

Science operates at many levels. For some, it is sufficient that scientific theories be clever, beautiful, and inspirational. Others require that any science eventually make contact with compelling applications and be subjected to empirical evaluation. Without personally weighing in on this emotional debate, I note that in his 2004 presidential address at the Second World Congress of the Game Theory Society,¹⁷ Kalai reprised the three stages of any science as discussed by von Neumann and Morgenstern:

[W]hat is important is the gradual development of a theory, based on a careful analysis of the ordinary everyday interpretation of economic facts. The theory finally developed must be mathematically rigorous and conceptually general. Its first applications are necessarily to elementary problems where the result has never been in doubt and no theory is actually required. At this early stage the application serves to corroborate the theory. The next stage develops when the theory is applied to somewhat more complicated situations in which

it may already lead to a certain extent beyond the obvious and the familiar. Here theory and application corroborate each other mutually. Beyond this lies the field of real success: genuine predictions by theory. It is well known that all mathematized sciences have gone through these successive phases of evolution.⁴⁰

So at least von Neumann, the father of modern-day game theory and computer science, attached importance to spanning the spectrum from the theoretical to the applied. Pragmatics may be critical to achieving von Neumann and Morgenstern’s third stage, and it could propel a joint endeavor between computer science and game theory. □

Acknowledgments

I thank Alon Altman, Joe Halpern, Sam Jeong, Daphne Koller, Tim Roughgarden, Moshe Vardi, Mike Wellman, and anonymous referees for their useful help and suggestions. Of course, all errors, opinions, and erroneous opinions are mine alone.

References

1. Bratman, M.E. *Intention, Plans, and Practical Reason*. CSLI Publications, Stanford University, 1987.
2. Chen, X. and Deng, X. Settling the complexity of 2-player Nash-equilibrium. *FoCS*, 2006.
3. Cohen, P.R. and Levesque, H.J. Intention is choice with commitment. *Artificial Intelligence* 42, 2–3 (1990), 213–261.
4. Conitzer, V. and Sandholm, T. Complexity results about Nash equilibria. *IJCAI*, 2003, 761–771.
5. Conitzer, V. and Sandholm, T. Computing Shapley values, manipulating value division schemes, and checking core membership in multi-issue domains. *AAAI*, 2004.
6. Cramton, P.C., Shoham, Y. and Steinberg, R. (eds). *Combinatorial Auctions*. MIT Press, 2006.
7. Deng, X. and Papadimitriou, C.H. On the complexity of cooperative solution concepts. *Mathematics of Operations Research* 19, 257, 1994.
8. Fagin, R., Halpern, J.Y., Moses, Y. and Vardi, M.Y. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995.
9. Niemela, I., Brewka, G., and Truszczyński, M. Nonmonotonic reasoning. In *Handbook of Knowledge Representation*. F. van Harmelen, V. Lifschitz, and B. Porter. (eds.), Elsevier, 2007.
10. Gilboa, I. and Zemel, E. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 80–93, 1989.
11. Govindan, S. and Wilson, R. A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 2003.
12. Greenwald, A. and Littman, M.L. (eds.). Special issue on learning and computational game theory. *Machine Learning* 67, 1–2, 2007.
13. Halpern, J.Y. A computer scientist looks at game theory. *Games and Economic Behavior* 45, 1 (2003), 114–132.
14. Halpern, J.Y. Computer science and game theory: A brief survey. In *The New Palgrave Dictionary of Economics*. S.N. Durlauf and L.E. Blume (eds.), Palgrave MacMillan, 2008.
15. Jeong, S. and Shoham, Y. Marginal Contribution Nets: A compact representation scheme for coalitional games. In *Proceedings of the 6th ACM Conference on Electronic Commerce*, 2005.
16. Kalai, E. Games, computers, and O.R. In *ACM/SIAM Symposium on Discrete Algorithms*, 1995.
17. Kalai, E. Presidential address. The Second World Congress of the Game Theory Society,

Marseille, 2004. *Games and Economic Behavior*, P. Reny (ed.), in press.

18. Kearns, M.J., Littman, M.L., and Singh, S.P. Graphical models for game theory. *UAI*, 2001.
19. Koller, D. and Milch, B. Multiagent influence diagrams for representing and solving games. *IJCAI*, 2001.
20. LaMura, P. Game networks. *UAI*, 2000.
21. Leyton-Brown, K. and Tennenholtz, M. Local-effect games. *IJCAI*, 2003.
22. Linial, N. Game theoretic aspects of computing. In *Handbook of Game Theory*, R.J. Aumann and S. Hart (eds.), 1339–1395, Elsevier Science, 1994.
23. Minsky, M. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. New York: Simon and Shuster, 2007.
24. Neyman, A. Bounded complexity justifies cooperation in finitely repeated prisoner’s dilemma. *Economic Letters*, 1985, 227–229.
25. Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. (eds). *Algorithmic Game Theory*. Cambridge University Press, 2007.
26. Nisan, N. and Ronen, A. Algorithmic mechanism design. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, 1999, 129–140.
27. Papadimitriou, C.H. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences* 48, 3 (1004), 498–532.
28. Papadimitriou, C.H. and Yannakakis, M. On bounded rationality and computational complexity. In *Proceedings of the Symposium on the Theory of Computing*, 1994, 726–733.
29. Papadimitriou, C.H. Algorithms, games, and the Internet. *Lecture Notes in Computer Science* 2076, 2001.
30. Pappas, P. Belief revision. In *Handbook of Knowledge Representation*. F. van Harmelen, V. Lifschitz, and B. Porter (eds.). Elsevier, 2007.
31. Porter, R., Nudelman, E., and Shoham, Y. Simple search methods for finding a Nash equilibrium. In *Proceedings of the National Conference on Artificial Intelligence*, 2004, 664–669.
32. Roughgarden, T. Computing equilibria: A computational complexity perspective. *Economic Theory*, 2008.
33. Rubinstein, A. *Modeling Bounded Rationality*. MIT Press, Cambridge, MA, 1998.
34. Savage, L.J. *The Foundations of Statistics*. John Wiley and Sons, NY, 1954. (Second Edition: Dover Press, 1972).
35. Shoham, Y. and Leyton-Brown, K. *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press, 2008.
36. Tennenholtz, M. Program equilibrium. *Games and Economic Behavior* 49, 2004, 363–373.
37. van Benthem, J. *Exploring Logical Dynamics*. Center for the Study of Language and Information, Stanford, CA, 1997.
38. van Benthem, J. When are two games the same? In *LOFT-III*, 1998 (ILLC preprint, 1999).
39. Vohra, R. and Wellman, M.P. (eds.). Special issue on foundations of multiagent learning. *Artificial Intelligence* 171, 1 (2007).
40. von Neumann, J. and Morgenstern, O. *Theory of Games and Economic Behavior*. Second Edition. Princeton University Press, 1947.
41. von Stengel, B. Computing equilibria for two-person games. In *Handbook of Game Theory*, vol. III, chap. 45. R. Aumann and S. Hart (eds.). Elsevier, Amsterdam, 2002, 1723–1759.
42. Wellman, M.P., Greenwald, A., and Stone, P. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. MIT Press, 2007.
43. Zinkevich, M., Bowling, M., and Burch, N. A new algorithm for generating equilibria in massive zero-sum games. In *Proceedings of the 22nd Conference on AI*, 2007, 788–793.

This work has been supported by NSF grant TR-0205633.

Yoav Shoham (<http://cs.stanford.edu/~shoham>) is a professor of computer science at Stanford University, Stanford, CA.

ICL 2008 Special Track Call for Papers

The 11th International Conference
"Interactive Computer-Aided Learning"
ICL2008 from September 24-26, 2008
in Villach, Austria has again a Special Track

Educational MashUps (EMA)

International Conference ICL

Mashups involve the reuse, or remixing, of works of art, of content, and/or of data for purposes that usually were not intended or even imagined by the original creators.

The term data mashup describes a Web site or application that combines the data and functionality of multiple Web sites into an integrated experience.

– Brian Lamb, University of British Columbia

Conference chair

M. Auer (Carinthia Tech
Institute Villach)

Special track chair

Linmi Tao (tao.linmi@gmail.com)
Tsinghua University Beijing, China

Proceedings

The proceedings will be published on CD in cooperation with the Kassel University Press (own ISBN number).

Submission of papers

Extended abstracts should be submitted using the **Electronic Submission System**.

The extended abstract should comprise up to two pages, informing the program committee about the aim of the approach (study, tool) reported, experiences gained and the form and result of evaluations conducted.

Proposals for tutorials and the exhibition also may be submitted in a short form to: info@icl-conference.org.

More information

<http://www.icl-conference.org> info@icl-conference.org

General information

The conference will be organized by the Carinthia Tech Institute in Villach. Conference venue is the Conference Center Villach.

Topics of interest

The goals of this special track are two-fold:

- 1 presenting novel fundamental techniques applicable to EMA, and
- 2 showcasing solutions to EMA applications.

Manuscripts are solicited to address a wide range of topics in EMA, including but not limited to the following:

- General framework for EMA
- Open and discoverable educational resources for reusing and remixing
- Novel technologies and APIs for EMA
- Web site or application that combines the data and functionality of multiple Web sites into an integrated experience in education.
- Educational examples of Google Map Mashup and Yahoo! Pipes interface

Types of contributions

- Full Papers: 20 minutes presentation followed by a panel discussion
- Interactive Demonstrations: 15 minutes demonstration.
- Poster Presentations

Other opportunities to participate:

- Exhibit at the ICL products and developments of e-learning.

P. 82

**Technical
Perspective**
**A Methodology for
Evaluating Computer
System Performance**

By William Pugh

P. 83

Wake Up and Smell the Coffee: Evaluation Methodology for the 21st Century

By Stephen M. Blackburn, Kathryn S. McKinley, Robin Garner, Chris Hoffmann, Asjad M. Khan, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot, B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann

P. 90

**Technical
Perspective**
**Transactions
are Tomorrow's
Loads and Stores**

By Nir Shavit

P. 91

Composable Memory Transactions

By Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy

Technical Perspective

A Methodology for Evaluating Computer System Performance

By William Pugh

COMPUTER SCIENCE HAS long had a solid foundation for evaluating the performance of algorithms. The asymptotic complexity of the time required by an algorithm is well defined and usually tractable, allowing for a clear evaluation of whether one algorithm provides a fundamental improvement over another. More nuanced and alternative evaluations, such as amortized and randomized analysis, provide additional insights into the fundamental advantages of different algorithms.

Unfortunately, the situation is even grimmer when evaluating the performance of a computer system, whether that system is a computer architecture, a compiler, a graphics processor, or a runtime system. Given a specific application, it is often fairly straightforward to execute the application on various systems and evaluate which system offers faster execution of that application on the provided input. Of course, once an application has been run on a particular input, one generally does not need to rerun it on that same input.

What programmers really want is some way to evaluate which system is likely to provide better performance on applications and data sets run in the future, thus making it the “better” system. Benchmarks also provide a way to examine how various system components behave and interact under load. Benchmarks should give repeatable results, even when rerun by an independent researcher or testing organization. A benchmark can be either a real or a synthetic application.

A synthetic application doesn’t compute anything useful but is designed to have performance characteristics that are representative of a range of real applications.

Benchmarks have an established history in computer science. The first widely used synthetic benchmark was the Whetstone benchmark written in

the Algol60 programming language in 1972 and later translated into many other languages. Several benchmarks became well known and widely used in research or commercial settings, or both. Examples include the Livermore loops, the Dhrystone benchmark, the Linpack benchmark, and the Perfect Club benchmarks.

The design and selection of benchmarks, however, has traditionally been a matter of art and taste, as much science as engineering. The paper here by the DaCapo team is the best effort I’ve seen in providing a sound basis for selecting benchmarks. Historically, there has not been any standard methodology for deciding whether or not a benchmark did indeed provide a representative measure of a system’s performance within a particular domain. A more serious problem with benchmarks is that they age poorly. Benchmarks often do a reasonable job of evaluating the performance of applications at the time they are proposed. However, three things tend to make benchmarks grow less useful over time:

- ▶ As machines and memories grow faster and larger, the sizes of application data sets grow as well. What was considered a reasonable problem size when a benchmark was proposed soon becomes a trivial example that fits in the on-processor cache.

- ▶ The actual applications that people use systems for evolve over time, and benchmarks that were once representative become less so.

- ▶ The weight attached to benchmark performance encourages developers of computer systems to optimize, tune, and tweak their systems in ways that improve their performance on the benchmarks but not more generally, making the benchmarks—again—less representative.

Almost every systems researcher and commercial software developer has a personal horror story about a poorly designed benchmark that was

difficult to use, produced misleading results, or focused attention on the wrong problem for too long. One such story in my own experience involves the SPEC JVM98 db benchmark intended to represent a database benchmark. Several early papers on removing redundant or useless synchronization from Java programs focused on this benchmark, since removing such synchronization could produce a 20% to 30% speed improvement in the benchmark. However, closer examination revealed that more than 70% of the CPU time for this benchmark was spent in a badly written 20-line Shell sort; replacing the handwritten sort with a call to the built-in sort function doubled the execution speed, even without removing the useless synchronization.

The DaCapo research group offers what seems to be an exceptionally well engineered set of benchmarks for evaluating Java computer systems. This includes not only selecting the benchmark applications, but designing a substantial infrastructure to support the execution and evaluation of benchmark executions.

Far more important than the actual selection of the benchmarks and the engineering infrastructure, the DaCapo team has put together an excellent description of best practices for using benchmarks to evaluate Java system performance, as well as a principled approach for evaluating whether a suite of benchmark applications is, in fact, sufficiently diverse. This approach involves measuring a number of characteristics of each application, and then applying principal component analysis (PCA) to determine whether the applications do have fundamental differences, or if they basically measure the same aspects of a system. I hope the methodology described in the paper will allow the DaCapo benchmark suite, and others, to be evaluated so they can evolve in ways that make them useful as well as meaningful for more than just a moment in time. □

William Pugh (pugh@cs.umd.edu) is a professor in the Department of Computer Science at the University of Maryland, College Park.

Wake Up and Smell the Coffee: Evaluation Methodology for the 21st Century

By Stephen M. Blackburn, Kathryn S. McKinley, Robin Garner, Chris Hoffmann, Asjad M. Khan, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot, B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann

Abstract

Evaluation methodology underpins all innovation in experimental computer science. It requires relevant *workloads*, appropriate *experimental design*, and *rigorous analysis*. Unfortunately, methodology is not keeping pace with the changes in our field. The rise of managed languages such as Java, C#, and Ruby in the past decade and the imminent rise of commodity multicore architectures for the next decade pose new methodological challenges that are not yet widely understood. This paper explores the consequences of our collective inattention to methodology on innovation, makes recommendations for addressing this problem in one domain, and provides guidelines for other domains. We describe benchmark suite design, experimental design, and analysis for evaluating Java applications. For example, we introduce new criteria for measuring and selecting diverse applications for a benchmark suite. We show that the complexity and nondeterminism of the Java runtime system make experimental design a first-order consideration, and we recommend mechanisms for addressing complexity and nondeterminism. Drawing on these results, we suggest how to adapt methodology more broadly. To continue to deliver innovations, our field needs to significantly increase participation in and funding for developing sound methodological foundations.

1. INTRODUCTION

Methodology is the foundation for judging innovation in experimental computer science. It therefore directs and misdirects our research. Flawed methodology can make good ideas look bad or bad ideas look good. Like any infrastructure, such as bridges and power lines, methodology is often mundane and thus vulnerable to neglect. While systemic misdirection of research is not as dramatic as a bridge collapse¹¹ or complete power failure,¹⁰ the scientific and economic cost may be considerable. Sound methodology includes using appropriate workloads, principled experimental design, and rigorous analysis. Unfortunately, many of us struggle to adapt to the rapidly changing computer science landscape. We use archaic benchmarks, outdated experimental designs, and/or inadequate data analysis. This paper explores the methodological gap, its consequences, and some solutions. We use the commercial uptake of managed languages over the past decade as the driving example.

Many developers today choose managed languages, which provide: (1) memory and type safety, (2) automatic memory management, (3) dynamic code execution, and (4) well-defined boundaries between type-safe and unsafe code (e.g., JNI and Pinvoke). Many such languages are also object-oriented. Managed languages include Java, C#, Python, and Ruby. C and C++ are not managed languages; they are compiled-ahead-of-time, not garbage collected, and unsafe. Unfortunately, managed languages add at least three new degrees of freedom to experimental evaluation: (1) a *space-time trade-off* due to garbage collection, in which heap size is a control variable, (2) *nondeterminism* due to adaptive optimization and sampling technologies, and (3) system *warm-up* due to dynamic class loading and just-in-time (JIT) compilation.

Although programming language researchers have embraced managed languages, many have not evolved their evaluation methodologies to address these additional degrees of freedom. As we shall show, weak methodology leads to incorrect findings. Equally problematic, most architecture and operating systems researchers do not use appropriate workloads. Most ignore managed languages entirely, despite their commercial prominence. They continue to use C and C++ benchmarks, perhaps because of the significant cost and challenges of developing expertise in new infrastructure. Regardless of the reasons, the current state of methodology for managed languages often provides bad results or no results.

To combat this neglect, computer scientists must be vigilant in their methodology. This paper describes how we addressed some of these problems for Java and makes recommendations for other domains. We discuss how benchmark designers can create *forward-looking and diverse workloads* and how researchers should use them. We then present a set of *experimental design guidelines* that accommodate complex and nondeterministic workloads. We show that managed languages make it much harder to produce meaningful results and suggest how to identify and explore control variables. Finally, we discuss the importance of *rigorous analysis*⁸ for complex nondeterministic systems that are not amenable to trivial empirical methods.

We address neglect in one domain, at one point in time, but the broader problem is widespread and growing. For example, researchers and industry are pouring resources into and exploring new approaches for embedded systems, multicore architectures, and concurrent programming models. However, without consequent investments

in methodology, how can we confidently evaluate these approaches? The community must take responsibility for methodology. For example, many Java evaluations still use SPECjvm98, which is badly out of date. Out-of-date benchmarks are problematic because they pose last year's problems and can lead to different conclusions.¹⁷ To ensure a solid foundation for future innovation, the community must make continuous and substantial investments. Establishing community standards and sustaining these investments require open software infrastructures containing the consequent artifacts.

For our part, we developed a new benchmark suite and new methodologies. We estimate that we have spent 10,000 person-hours to date developing the DaCapo suite and associated infrastructure, none of it directly funded. Such a major undertaking would be impossible without a large number of contributing institutions and individuals. Just as NSF and DARPA have invested in networking infrastructure to foster the past and future generations of the Internet, our community needs foundational investment in methodological infrastructure to build next-generation applications, software systems, and architectures. Without this investment, what will be the cost to researchers, industry, and society in lost opportunities?

2. WORKLOAD DESIGN AND USE

The DaCapo research group embarked on building a Java benchmark suite in 2003 after we highlighted the dearth of realistic Java benchmarks to an NSF review panel. The panel suggested we solve our own problem, but our grant was for dynamic optimizations. NSF did not provide additional funds for benchmark development, but we forged ahead regardless. The standard workloads at the time, SPECjvm98 and SPECjbb2000,^{14,15} were out of date. For example, SPECjvm98 and SPECjbb2000 make meager use of Java language features, and SPECjvm98 has a tiny code and memory footprint. (SPEC measurements are in a technical report³.) We therefore set out to create a suite suitable for research, a goal that adds new requirements beyond SPEC's goal of product comparisons. Our goals were:

Relevant and diverse workload: A diverse, widely used set of nontrivial applications that provide a compelling platform for innovation.

Suitable for research: A controlled, tractable workload amenable to analysis and experiments.

We selected the following benchmarks for the initial release of the DaCapo suite, based on criteria described below.

<i>antlr</i>	Parser generator and translator generator
<i>bloat</i>	Java bytecode-level optimization and analysis tool
<i>chart</i>	Graph-plotting toolkit and PDF renderer
<i>eclipse</i>	Integrated development environment (IDE)
<i>fop</i>	Output-device-independent print formatter
<i>hsqldb</i>	SQL relational database engine written in Java
<i>jython</i>	Python interpreter written in Java
<i>lindex</i>	Text-indexing tool
<i>lusearch</i>	Text-search tool

<i>pmd</i>	Source code analyzer for Java
<i>xalan</i>	XSLT transformer for XML documents

2.1. Relevance and diversity

No workload is definitive, but a narrow scope makes it possible to attain some coverage. We limited the DaCapo suite to nontrivial, actively maintained real-world Java applications. We solicited and collected candidate applications. Because source code supports research, we considered only open-source applications. We first packaged candidates into a prototype DaCapo harness and tuned them with inputs that produced *tractable* execution times suitable for experimentation, that is, around a minute on 2006 commodity hardware. Section 2.2 describes how the DaCapo packaging provides tractability and standardization.

We then quantitatively and qualitatively evaluated each candidate. Table 1 lists the static and dynamic metrics we used to ensure that the benchmarks were relevant and diverse. Our original paper⁴ presents the DaCapo metric data and our companion technical report³ adds SPECjvm98 and SPECjbb200. We compared against SPEC as a reference point and compared candidates with each other to ensure diversity.

We used new and standard metrics. Our standard metrics included the static CK metrics, which measure code complexity of object-oriented programs⁶; dynamic heap composition graphs, which measure time-varying lifetime properties of the heap¹⁶; and architectural characteristics such as branch misprediction rates and instruction mix. We introduced new metrics to capture domain-specific characteristics of Java such as allocation rate, ratio of

Table 1: Quantitative selection metrics.

Metric	Description
Code Metrics	
CK metrics ⁶	Object-oriented programming metrics measuring source code complexity
Code size	Numbers of classes loaded, methods declared, total bytecodes compiled
Code footprint	Instruction cache and I-TLB misses
Optimization	Number of methods compiled, number optimized, percentage hot
Heap Metrics	
Allocation	Total bytes/objects allocated, average object size
Heap footprint	Maximum live bytes/objects, nursery survival rate
Fan-out/fan-in	Mean incoming and outgoing pointers per object
Pointer distance	Mean distance in bytes of each pointer encountered in a snapshot traversal of an age-ordered heap
Mutation distance	Mean distance in bytes of each pointer dynamically created/mutated by the application in an age-ordered heap
Architecture Metrics	
Instruction mix	Mix of branches, ALU, and memory instructions
Branches	Branch mispredictions per instruction for PMM predictor
Register dependence	Register dependence distances

allocated to live memory, and heap mutation rate. These new metrics included summaries and time series of allocated and live object size demographics, summaries and time series of pointer distances, and summaries and time series of mutation distances. Pointer distance and mutation distance time-series metrics summarize the lengths of the edges that form the application's object graph. We designed these metrics and their means of collection to be abstract, so that the measurements are VM-neutral.⁴

Figure 1 qualitatively illustrates the temporal complexity of heap composition and pointer distance metrics for two benchmarks, *_209_db* and *eclipse*. With respect to our metrics, *eclipse* from DaCapo is qualitatively richer than *_209_db* from SPECjvm98. Our original paper explains how to read these graphs and includes dozens of graphs, representing mountains of data.⁴ Furthermore, it shows that the DaCapo benchmarks substantially improve over SPECjvm98 on all measured metrics. To confirm the diversity of the suite, we applied principal component analysis (PCA)⁷ to the summary metrics. PCA is a multivariate statistical technique for reducing a large N -dimensional space into a lower-dimensional uncorrelated space. If the benchmarks are uncorrelated in lower-dimensional space, then they are also uncorrelated in the higher-dimensional space. The analysis shows that the DaCapo benchmarks are diverse, nontrivial real-world applications with significant memory load, code complexity, and code size.

Because the applications come from active projects, they include unresolved performance anomalies, both typical and unusual programming idioms, and bugs. Although not our intention, their rich use of Java features uncovered bugs in some commercial JVMs. The suite notably omits Java application servers, embedded Java applications, and numerically intensive applications. Only a few benchmarks are explicitly concurrent. To remain relevant, we plan to update the DaCapo benchmarks every two years to their latest version, add new applications, and delete applications that have become less relevant. This relatively tight schedule should reduce the extent to which vendors may tune their products to the benchmarks (which is standard practice, notably for SPECjbb2000¹).

As far as we know, we are the first to use quantitative metrics and PCA analysis to ensure that our suite is diverse and nontrivial. The designers of future suites should choose additional aggregate and time-varying metrics that directly address the domain of interest. For example, metrics for concurrent or embedded applications might include a measure of the fraction of time spent executing purely sequential code, maximum and time-varying degree of parallelism, and a measure of sharing between threads.

2.2. Suitable for research

We decided that making the benchmarks tractable, standardized, and suitable for research was a high priority. While not technically deep, good packaging is extremely time consuming and affects usability. Researchers need tractable workloads because they often run thousands of executions for a single experiment. Consider comparing four garbage collectors over 16 heap sizes—that is, we need 64

combinations to measure. Teasing apart the performance differences with multiple hardware performance monitors may add eight or more differently instrumented runs per combination. Using five trials to ensure statistical significance requires a grand total of 2560 test runs. If a single benchmark test run takes as long as 20 min (the time limit is 30 min on SPECjbb¹⁵), we would need over a month on one machine for just one benchmark comparison—and surely we should test the four garbage collectors on many benchmarks, not just one.

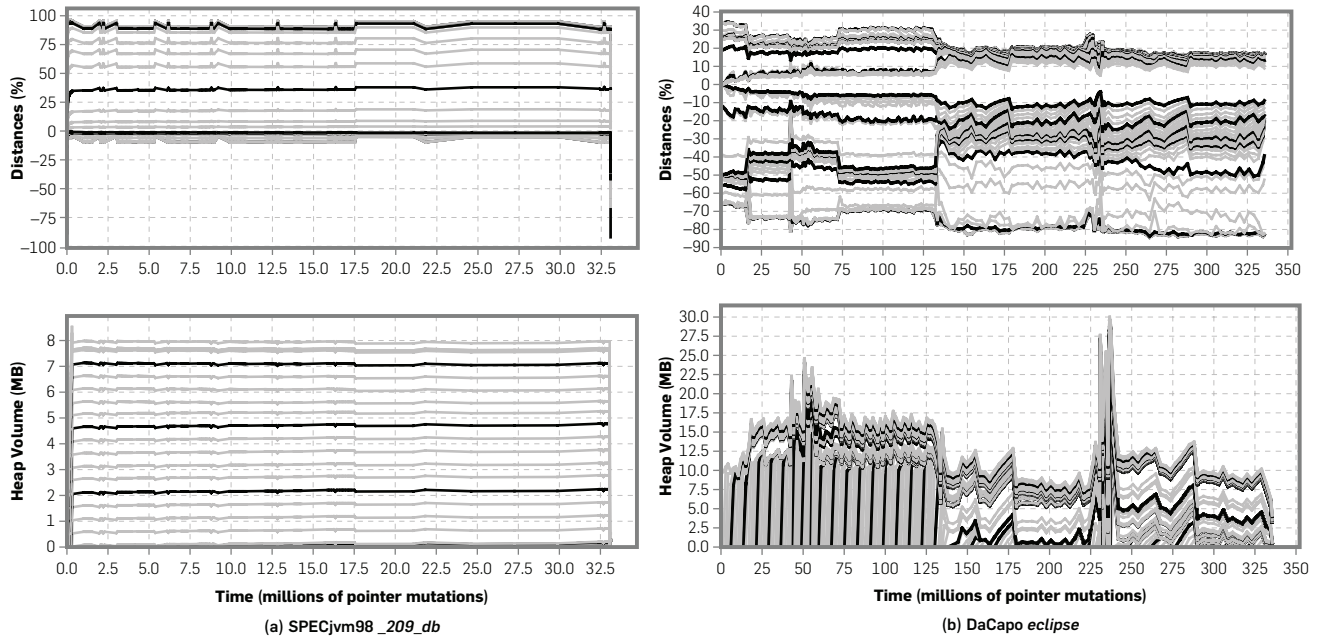
Moreover, time-limited workloads do not hold work constant, so they are analytically inconvenient for reproducibility and controlling load on the JIT compiler and the garbage collector. Cycle-accurate simulation, which slows execution down by orders of magnitude, further amplifies the need for tractability. We therefore provide work-limited benchmarks with three input sizes: small, default, and large. For some of the benchmarks, large and default are the same. The largest ones typically executed in around a minute on circa 2006 commodity high-performance architectures.

We make simplicity our priority for packaging; we ship the suite as a single self-contained Java jar file. The file contains all benchmarks, a harness, input data, and checksums for correctness. The harness checksums the output of each iteration and compares it to a stored value. If the values do not match, the benchmark fails. We provide extensive configuration options for specifying the number of iterations, the ability to run to convergence with customized convergence criteria, and callback hooks before and after every iteration. For example, the user-defined callbacks can turn hardware performance counters on and off, or switch a simulator in and out of detailed simulation mode. We use these features extensively and are heartened to see others using them.¹² For standardization and analytical clarity, our benchmarks require only a single host and we avoid components that require user configuration. By contrast, SPEC jAppServer, which models real-world application servers, requires multiple hosts and depends on third-party-configurable components such as a database. Here we traded some relevance for control and analytical clarity.

We provide a separate “source” jar to build the entire suite from scratch. For licensing reasons, the source jar automatically downloads the Java code from the licensor. With assistance from our users,⁵ our packaging now facilitates static whole program analysis, which is not required for standard Java implementations. Since the entire suite and harness are open-source, we happily accept contributions from our users.

2.3. The researcher

Appropriate workload selection is a task for the community, consortia, the workload designer, and the researcher. Researchers make a workload selection, either implicitly or explicitly, when they conduct an experiment. This selection is often automatic: “Let’s use the same thing we used last time!” Since researchers invest heavily in their evaluation methodology and infrastructure, this path offers the least resistance. Instead, we need to identify the workloads and methodologies that best serve the research evaluation. If

Figure 1: Two time-varying selection metrics. Pointer distance (top) and heap composition (bottom) as a function of time.

there is no satisfactory answer, it is time to form or join a consortium and create new suitable workloads and supporting infrastructure.

Do Not Cherry-Pick! A well-designed benchmark suite reflects a range of behaviors and should be used as a whole. Perez et al. demonstrate with alarming clarity that cherry-picking changes the results of performance evaluation.¹³ They simulate 12 previously published cache architecture optimizations in an apples-to-apples evaluation on a suite of 26 SPECcpu benchmarks. There is one clear winner with all 26 benchmarks. There is a choice of 2 different winners with a suitable subset of 23 benchmarks, 6 winners with subsets of 18, and 11 winners with 7. When methodology allows researchers a choice among 11 winners from 12 candidates, the risk of incorrect conclusions, by either mischief or error, is too high. Section 3.1 shows that Java is equally vulnerable to subsetting.

Run every benchmark. If it is impossible to report results for every benchmark because of space or time constraints, bugs, or relevance, explain why. For example, if you are proposing an optimization for multithreaded Java workloads, you may wish to exclude benchmarks that do not exhibit concurrency. In this case, we recommend reporting all the results but highlighting the most pertinent. Otherwise readers are left guessing as to the impact of the “optimization” on the omitted workloads—with key data omitted, readers and reviewers should not give researchers the benefit of the doubt.

3. EXPERIMENTAL DESIGN

Sound experimental design requires a meaningful baseline and comparisons that control key parameters. Most researchers choose and justify a baseline well, but identifying which parameters to control and how to control them is challenging.

3.1. Gaming your results

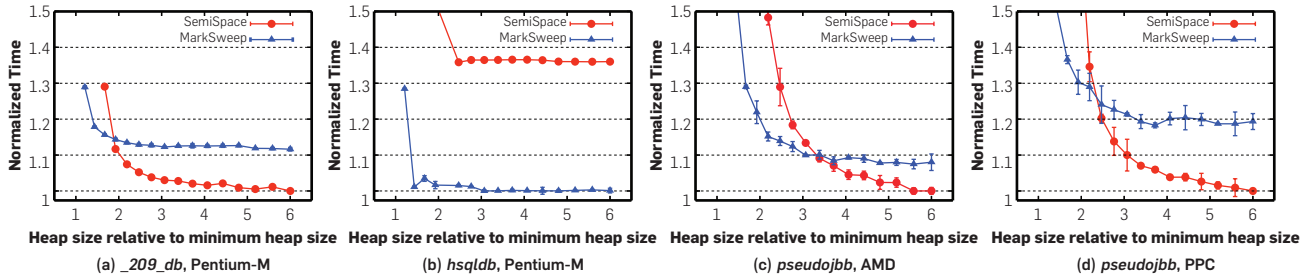
The complexity and degrees of freedom inherent in these systems make it easy to produce misleading results through errors, omissions, or mischief. Figure 2 presents four results from a detailed comparison of two garbage collectors. The JVM, architecture, and other evaluation details appear in the original paper.⁴ More garbage collector implementation details are in Blackburn et al.² Each graph shows normalized time (lower is better) across a range of heap sizes that expose the space–time tradeoff for implementations of two canonical garbage collector designs, SemiSpace and MarkSweep.

Subsetting Figure 2 badly misleads us in at least three ways: (1) Figure 2(c) shows that by selecting a single heap size rather than plotting a continuum, the results can produce diametrically opposite conclusions. At $2.1 \times$ maximum heap size, MarkSweep performs much better than SemiSpace, while at $6.0 \times$ maximum heap size, SemiSpace performs better. Figures 2(a) and 2(d) exhibit this same dichotomy, but have different crossover points. Unfortunately, some researchers are still evaluating the performance of garbage-collected languages *without* varying heap size. (2) Figures 2(a) and 2(b) confirm the need to use an entire benchmark suite. Although *_209_db* and *hsqldb* are established in-memory database benchmarks, SemiSpace performs better for *_209_db* in large heaps, while MarkSweep is always better for *hsqldb*. (3) Figures 2(c) and 2(d) show that the architecture significantly impacts conclusions at these heap size ranges. MarkSweep is better at more heap sizes for AMD hardware as shown in Figure 2(c). However, Figure 2(d) shows SemiSpace is better at more heap sizes for PowerPC (PPC) hardware. This example of garbage-collection evaluation illustrates a small subset of the pitfalls in evaluating the performance of managed languages.

3.2. Control in a changing world

Understanding what to control and how to control it in an

Figure 2: Gaming your results. Four ways to compare two garbage collectors.



experimental system is clearly important. For a classic comparison of Fortran, C, or C++ systems, there are at least two degrees of freedom to control: (a) the *host platform* (hardware and operating system) and (b) the *language runtime* (compiler and associated libraries). Over the years, researchers have evolved solid methodologies for evaluating compiler, library, and architectural enhancements that target these languages. Consider a compiler optimization for improving cache locality. Accepted practice is to compile with and without the optimization and report how often the compiler applied the optimization. To eliminate interference from other processes, one runs the versions standalone on one or more architectures and measures miss rates with either performance counters or a simulator. This methodology evolved, but is now extremely familiar. Once researchers invest in a methodology, the challenge is to notice when the world has changed, and to figure out *how* to adapt.

Modern managed runtimes such as Java add at least three more degrees of freedom: (c) *heap size*, (d) *nondeterminism*, and (e) *warm-up* of the runtime system.

Heap Size: Managed languages use garbage collection to detect unreachable objects, rather than relying on the programmer to explicitly delete objects. Garbage collection is fundamentally a space-time trade-off between the efficacy of space reclamation and time spent reclaiming objects; heap size is the key control variable. The smaller the heap size, the more often the garbage collector will be invoked and the more work it will perform.

Nondeterminism: Deterministic profiling metrics are expensive. High-performance JVMs therefore use approximate execution frequencies computed by low-overhead dynamic sampling to select which methods the JIT compiler will optimize and how. For example, a method may happen to be sampled N times in one invocation and $N + 3$ in another; if the optimizer uses a hot-method threshold of $N + 1$, it will make different choices. Due to this nondeterminism, code quality usually does not reach the *same* steady state on a deterministic workload across independent JVM invocations.

Warm-Up: A single invocation of the JVM will often execute the same application repeatedly. The first iteration of the application usually includes the largest amount of dynamic compilation. Later iterations usually have both less compilation and better application code quality. Eventually, code quality may reach a steady state. Code quality thus “warms up.” Steady state is the most frequent use-case. For example, application servers run their code many times in the same

JVM invocation and thus care most about steady-state performance. Controlling for code warm-up is an important aspect of experimental design for high-performance runtimes.

3.3. Case study

We consider performance evaluation of a new garbage collector as an example of experimental design. We describe the context and then show how to control the factors described above to produce a sound experimental design.

Two key context-specific factors for garbage-collection evaluation are (a) the *space-time trade-off* as discussed above and (b) the relationship between the collector and *mutator* (the term for the application itself in the garbage-collection literature). For simplicity, we consider a *stop-the-world* garbage collector, in which the collector and the mutator never overlap in execution. This separation eases measurement of the mutator and collector. Some collector-specific code mixes with the mutator: object allocation and *write barriers*, which identify pointers that cross between independently collected regions. This code impacts both the mutator and the JIT compiler. Furthermore, the collector greatly affects mutator locality, due to the allocation policy and any movement of objects at collection time.

Meaningful Baseline: Comparing against the state of the art is ideal, but practical only when researchers make their implementations publicly available. Researchers can then implement their approaches using the same tools or control for infrastructure differences to make apples-to-apples comparisons. Garbage-collection evaluations often use generational MarkSweep collectors as a baseline because these collectors are widely used in high-performance VMs and perform well.

Host Platform: Garbage collectors exhibit architecture-dependent performance properties that are best revealed with an evaluation across multiple architectures, as shown in Figures 2(c) and 2(d). These properties include locality, the cost of write barriers, and the cost of synchronization instructions.

Language Runtime: The language runtime, libraries, and JIT compiler directly affect memory load, and so should be controlled. Implementing various collectors in a common toolkit factors out common shared mechanisms and focuses the comparison on the algorithmic differences between the collectors.

Heap Size: Garbage-collection evaluations should com-

pare performance across a range of benchmark-specific relative heap sizes, starting at the smallest heap in which any of the measured collectors can run, as shown in Figure 2. Each evaluated system must experience the same memory load, which requires forcing collections between iterations to normalize the heap and controlling the JIT compiler.

Nondeterminism: Nondeterministic JIT optimization plans lead to nondeterministic mutator performance. JIT optimization of collector-specific code, optimizations that elide allocations, and the fraction of time spent in collection may affect mutator behavior in ways that cannot be predicted or repeated. For example, in Jikes RVM, a Java-in-Java VM widely used by researchers, JIT compiler activity directly generates garbage collection load because the compiler allocates and executes in the same heap as the application. These effects make nondeterminism even more acute.

Warm-Up: For multi-iteration experiments, as the system warms up, mutator speeds increase, and JIT compiler activity decreases, the *fraction* of time spent in collection typically grows. Steady-state execution therefore accentuates the impact of the garbage collector as compared to start-up. Furthermore, the relative impact of collector-specific code will change as the code is more aggressively optimized. Evaluations must therefore control for code quality and warm-up.

3.4. Controlling nondeterminism

Of the three new degrees of freedom outlined in Section 3.2, we find dealing with nondeterminism to be the most methodologically challenging. Over time, we have adopted and recommend three different strategies: (a) use deterministic *replay* of optimization plans, which requires JVM support; (b) take multiple measurements in a single JVM invocation, after reaching steady state and turning off the JIT compiler; and (c) generate sufficient data points and apply suitable statistical analysis.⁸ Depending on the experiment, the researcher will want to perform one, two, or all of these experiments. The first two reduce nondeterminism for analysis purposes by controlling its sources. Statistical analysis of results from (a) and (b) will reveal whether differences from the remaining nondeterminism are significant. The choice of (c) accommodates larger factors of nondeterminism (see Section 4) and may be more realistic, but requires significantly more data points, at the expense of other experiments.

Replay Compilation: Replay compilation collects profile data and a compilation plan from one or more training runs, forms an optimization plan, and then replays it in subsequent, independent timing invocations.⁹ This methodology deterministically applies the JIT compiler, but requires modifications to the JVM. It isolates the JIT compiler activity, since replay eagerly compiles to the plan's final optimization level instead of lazily relying on dynamic recompilation triggers. Researchers can measure the first iteration for deterministic characterization of start-up behavior. Replay also removes most profiling overheads associated with the adaptive optimization system, which is turned off. As far as we are aware, production JVMs do not support replay compilation.

Multi-Iteration Determinism: An alternative approach

that does not depend on runtime support is to run multiple measurement iterations of a benchmark in a single invocation, *after* the runtime has reached steady state. Unlike replay, this approach does not support deterministic measurement of warm-up. We use this approach when gathering data from multiple hardware performance counters, which requires multiple distinct measurements of the same system. We first perform $N - 1$ unmeasured iterations of a benchmark while the JIT compiler warms up the code. We then turn the JIT compiler off and execute the N th iteration unmeasured to drain any JIT work queues. We measure the next K iterations. On each iteration, we gather different performance counters of interest. Since the code quality has reached steady state, it should be a representative mix of optimized and unoptimized code. Since the JIT compiler is turned off, the variation between the subsequent iterations should be low. The variation can be measured and verified.

3.5. Experimental design in other settings

In each experimental setting, the relative influence of the degrees of freedom, and how to control them, will vary. For example, when evaluating a *new compiler optimization*, researchers should hold the garbage-collection activity constant to keep it from obscuring the effect of the optimization. Comparing on multiple architectures is best, but is limited by the compiler back-end. When evaluating a *new architecture*, vary the garbage-collection load and JIT compiler activity, since both have distinctive execution profiles. Since architecture evaluation often involves very expensive simulation, eliminating nondeterminism is particularly important.

4. ANALYSIS

Researchers use data analysis to identify and articulate the significance of experimental results. This task is more challenging when systems and their evaluation become more complex, and the sheer volume of results grows. The primary data analysis task is one of aggregation: (a) across repeated experiments to defeat experimental noise and (b) across diverse experiments to draw conclusions.

Aggregating data across repeated experiments is a standard technique for increasing confidence in a noisy environment.⁸ In the limit, this approach is in tension with *tractability*, because researchers have only finite resources. Reducing sources of nondeterminism with sound experimental design improves tractability. Since noise cannot be eliminated altogether, multiple trials are inevitably necessary. Researchers must aggregate data from multiple trials and provide evidence such as confidence intervals to reveal whether the findings are significant. Georges et al.⁸ use a survey to show that current practice lacks statistical rigor and explain the appropriate tests for comparing alternatives.

Section 2.3 exhorts researchers not to cherry-pick benchmarks. Still, researchers need to convey results from diverse experiments succinctly, which necessitates aggregation. We encourage researchers (a) to include complete results and (b) to use appropriate summaries. For example, using the geometric mean dampens the skewing effect of one excellent result. Although industrial benchmarks will often produce

a single aggregate score over a suite, this methodology is brittle because the result depends entirely on vagaries of the suite composition.¹⁸ For example, while it is tempting to cite your best result—“we outperform *X* by up to 1000%”—stating an aggregate together with the best and worst results is more honest and insightful.

5. CONCLUSION


Methodology plays a strategic role in experimental computer science research and development by creating a common ground for evaluating ideas and products. Sound methodology relies on *relevant workloads*, *principled experimental design*, and *rigorous analysis*. Evaluation methodology can therefore have a significant impact on a research field, potentially accelerating, retarding, or misdirecting energy and innovation. However, we work within a fast-changing environment and our methodologies must adapt to remain sound and relevant. Prompted by concerns among ourselves and others about the state of the art, we spent thousands of hours at eight institutions examining and addressing the problems of evaluating Java applications. The lack of direct funding, the perception that methodology is mundane, and the magnitude of the effort surely explain why these efforts are uncommon.

We address neglect of evaluation methodology concretely, in one domain at one point in time, and draw broader lessons for experimental computer science. The development and maintenance of the DaCapo benchmark suite and associated

methodology have brought some much-needed improvement to our evaluations and to our particular field. However, experimental computer science cannot expect the upkeep of its methodological foundations to fall to ad hoc volunteer efforts. We encourage stakeholders such as industry and granting agencies to be forward-looking and make a systemic commitment to stem methodological neglect. Invest in the foundations of our innovation.

Acknowledgments

We thank Andrew Appel, Randy Chow, Frans Kaashoek, and Bill Pugh, who encouraged this project at our three year NSF ITR review. We thank Mark Wegman, who initiated the public availability of Jikes RVM, and the developers of Jikes RVM. We gratefully acknowledge Fahad Gilani, who wrote the original version of the measurement infrastructure for his ANU Masters thesis; Xianglong Huang and Narendran Sachindran, who helped develop the replay methodology; and Jungwoo Ha and Magnus Gustafsson, who helped developed the multi-iteration replay methodology. We thank Tom Horn for his proofreading, and Guy Steele for his careful reading and suggestions.

This work was supported by NSF ITR CCR-0085792, CNS-0719966, NSF CCF-0429859, NSF EIA-0303609, DARPA F33615-03-C-4106, ARC DP0452011, ARC DP0666059, Intel, IBM, and Microsoft. Any opinions, findings and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors. 

References

- Adamson, A., Dagastine, D., and Sarne, S. SPECjbb2005—A year in the life of a benchmark. *2007 SPEC Benchmark Workshop*. SPEC, Jan. 2007.
- Blackburn, S.M., Cheng P., and McKinley, K.S. Myths and realities: The performance impact of garbage collection. *Proceedings of the ACM Conference on Measurement and Modeling Computer Systems*, pp. 25–36, New York, NY, June 2004.
- Blackburn, S.M., Garner, R., Hoffman, C., Khan, A.M., McKinley, K.S., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S.Z., Hirzel, M., Hosking, A., Jump, M., Lee, H., Moss, J.E.B., Phansalkar, A., Stefanović, D., VanDrunen, T., von Dincklage, D., and Wiedermann, B. The DaCapo benchmarks: Java benchmarking development and analysis (extended version). Technical Report TR-CS-06-01, Dept. of Computer Science, Australian National University, 2006. <http://www.dacapobench.org>.
- Blackburn, S.M., Garner, R., Hoffman, C., Khan, A.M., McKinley, K.S., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S.Z., Hirzel, M., Hosking, A., Jump, M., Lee, H., Moss, J.E.B., Phansalkar, A., Stefanović, D., VanDrunen, T., von Dincklage, D., and Wiedermann, B. The DaCapo benchmarks: Java benchmarking development and analysis. *ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 169–190, Oct. 2006.
- Bodden, E., Hendren, L., and Lhoták, O. A staged static program analysis to improve the performance of runtime monitoring. *21st European Conference on Object-Oriented Programming, July 30th–August 3rd 2007*, Berlin, Germany, number 4609 in Lecture Notes in Computer Science, pp. 525–549, Springer Verlag, 2007.
- Chidamber, S.R. and Kemerer, C.F. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.
- Duntelman, G.H. *Principal Components Analysis*. Sage Publications, Newbury Park, CA, USA, 1989.
- Georges, A., Buytaert, D., and Eeckhout, L. Statistically rigorous Java performance evaluation. *ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 57–76, Montreal, Quebec, Canada, 2007.
- Huang, X., Blackburn, S.M., McKinley, K.S., Moss, J.E.B., Wang Z., and Cheng P. The garbage collection advantage: Improving mutator locality. *ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 69–80, Vancouver, BC, 2004.
- Leyland, B. Auckland central business district supply failure. *Power Engineering Journal*, 12(3):109–114, 1998.
- National Transportation Safety Board. NTSB urges bridge owners to perform load capacity calculations before modifications; I-35W investigation continues. SB-08-02. <http://www.nts.gov/Pressrel/> 2008/080115.html, Jan. 2008.
- Neelakantam, N., Rajwar, R., Srinivas, S., Srinivasan, U., and Zilles, C. Hardware atomicity for reliable software speculation. *ACM/IEEE International Symposium on Computer Architecture*, pp. 174–185, ACM, New York, NY, USA, 2007.
- Perez, D.G., Mouchard, G., and Temam, O. MicroLib: A case for the quantitative comparison of micro-architecture mechanisms. *International Symposium on Microarchitecture*, pp. 43–54, Portland, OR, Dec. 2004.
- Standard Performance Evaluation Corporation. *SPECjvm98 Documentation*, release 1.03 edition, March 1999.
- Standard Performance Evaluation Corporation. *SPECjbb2000 (Java Business Benchmark) Documentation*, release 1.01 edition, 2001.
- Stefanović, D. *Properties of Age-Based Automatic Memory Reclamation Algorithms*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, Dec. 1998.
- Yi, J.J., Vandierendonck, H., Eeckhout, L., and Lilja, D.J. The exigency of benchmark and compiler drift: Designing tomorrow's processors with yesterday's tools. *International Symposium on Supercomputing*, pp. 75–86, Cairns, Queensland, Australia, July 2006.
- Yoo, R.M., Lee, H.-H. S., Lee, H., and K. Chow. Hierarchical means: Single number benchmarking with workload cluster analysis. *IISWC 2007. IEEE 10th International Symposium on Workload Characterization*, pp. 204–213, IEEE, 2007.

Stephen M. Blackburn, Robin Garner, Daniel Frampton, Australian National University

Kathryn S. McKinley, Aashish Phansalkar, Ben Wiedermann, Maria Jump, University of Texas at Austin

Chris Hoffmann, Asjad M. Khan, J Eliot B. Moss, University of Massachusetts, Amherst

Rotem Bentzur, Daniel Feinberg, Darko Stefanović, University of New Mexico

Amer Diwan, Daniel von Dincklage, University of Colorado

Samuel Z. Guyer, Tufts University

Martin Hirzel, IBM

Antony Hosking, Purdue University

Han Lee, Intel

Thomas VanDrunen, Wheaton College

© 2008 ACM 0001-0782/08/0800 \$5.00

Technical Perspective

Transactions are Tomorrow's Loads and Stores

By Nir Shavit

IN COMPUTER SCIENCE, when we say “time is money,” we typically refer to two types of time that determine the costs and benefits of a given computer program: the time it takes the program to run, and the time it takes to write and maintain it. There is a delicate trade-off between these two types of time: the faster we want a program to run, the more time we need to spend when writing and maintaining it, and vice versa.

Until very recently, it seemed this trade-off could be mostly ignored. The job of making programs run faster fell into the lap of the hardware architects, who continued to deliver advances in single CPU clock speed at a reliable pace. These reliable speed increases allowed software engineers and programming language designers to focus on adding software constructs that offered substantial reductions in the time it takes to write and maintain code. How was this done? The terms that come to mind are abstraction, modularity, and compositionality.

Unfortunately, as we have all heard, things are about to change dramatically. Moore's Law has not been repealed—each year more and more transistors are being fit into the same space—but CPU clock speeds can no longer be effectively increased. Instead, hardware designers have turned to multicore architectures, in which multiple computing cores are included on each processor chip. The switch to multicore architectures promises increased parallelism, but not increased single-thread performance. Even if this increased parallelism is delivered at a reliable pace, the hardware designers cannot by themselves deliver reliable increases in the speed at which programs run. This job will fall into the laps of software engineers.

The main tool for handling concurrency in today's programming languages are locks—software constructs that allow sequences of loads and stores

to access data in a mutually exclusive manner. Indeed, lock-based programs have been known to deliver amazing performance on multicore architectures. However, it is becoming clear that, while using locks will allow us to continue to reduce the time it takes programs to run, they will cause the time it takes us to write and maintain our programs to shoot back up.

The heart of the problem is, perhaps, that no one really knows how to organize and maintain large systems that rely on locking. Locks are not modular and do not compose, and the association between locks and data is established mostly by convention. Ultimately, the association exists only in the mind of the programmer, and may be documented only in comments.

A promising solution to this problem is the introduction of *atomic memory* transactions as a multicore programming abstraction. While transactions have been used for years in the database community, they are now being proposed as an equal partner to, and perhaps even a replacement for, the loads and stores we typically use in our programs. The idea is simple: encapsulate sequences of loads and stores within a transaction, with the guarantee that if any operation takes place, they all do, and that if they do, they appear to other threads to do so atomically, as one indivisible operation.

Work on the design of efficient *transactional memory* implementations has been proceeding in earnest. However, there is a missing element: a framework of transactional memory-based concurrency that would provide the modularity and compositionality necessary when designing and maintaining large-scale concurrent systems.

This is where the breakthrough work on composable memory transactions by Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy takes center stage. They have pro-

vided, for the first time, a concurrent language model and a set of constructs that allow true simplicity in transactional programming.

One source of difficulty they had to overcome was that transactions are optimistic: they are tried but may fail and have to be rolled back, so one cannot allow events with side effects, for example I/O, to take place within transactions.

The authors' solution was to use transactions within a purely declarative language, which, as it turns out, is a perfect match for transactions since the type system explicitly separates computations that may have side effects from effect-free ones.

Another big problem was that concurrent programming requires that threads await events by other threads. Waiting on a condition outside a transaction greatly limits what it can do, and waiting inside a transaction can get it stuck.

The authors solved the problem by introducing new transactional constructs, among them an elegant retry command that allows a transaction to effectively abort, then restart only after a potentially good event has increased the likelihood of the condition being met. Quite surprisingly, retry is a key factor in allowing sequences of transactional actions to be composed so they all take effect together.

The language structure together with the added transactional composition constructs provide a clean framework that allows transactions to compose without giving up their natural advantages over locks: concurrent programmers can program using atomic operations that span multiple objects in memory, without having to break abstraction barriers.

If multicore architectures are the way to continue improving the time it takes programs to run, then perhaps composable memory transactions are the way to improve the time it takes us to write and maintain them. □

Nir Shavit (shavir@cs.tau.ac.il) is a past program chair of the ACM's PODC and SPAA conferences and a recipient of the ACM's 2004 Gödel Prize in Theoretical Computer Science.

Composable Memory Transactions

By Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy

Abstract

Writing concurrent programs is notoriously difficult and is of increasing practical importance. A particular source of concern is that even correctly implemented concurrency abstractions cannot be composed together to form larger abstractions. In this paper we present a concurrency model, based on *transactional memory*, that offers far richer composition. All the usual benefits of transactional memory are present (e.g., freedom from low-level deadlock), but in addition we describe modular forms of *blocking* and *choice* that were inaccessible in earlier work.

1. INTRODUCTION

The free lunch is over.²⁵ We have been used to the idea that our programs will go faster when we buy a next-generation processor, but that time has passed. While that next-generation chip will have more CPUs, each individual CPU will be no faster than the previous year's model. If we want our programs to run faster, we must learn to write parallel programs.

Writing parallel programs is notoriously tricky. Mainstream lock-based abstractions are difficult to use and they make it hard to design computer systems that are reliable and scalable. Furthermore, systems built using locks are difficult to compose without knowing about their internals.

To address some of these difficulties, several researchers (including ourselves) have proposed building programming language features over *software transactional memory* (STM), which can perform groups of memory operations atomically.²³ Using transactional memory instead of locks brings well-known advantages: freedom from deadlock and priority inversion, automatic roll-back on exceptions or timeouts, and freedom from the tension between lock granularity and concurrency.

Early work on software transactional memory suffered several shortcomings. Firstly, it did not prevent transactional code from bypassing the STM interface and accessing data directly at the same time as it is being accessed within a transaction. Such conflicts can go undetected and prevent transactions executing atomically. Furthermore, early STM systems did not provide a convincing story for building operations that may block—for example, a shared work-queue supporting operations that wait if the queue becomes empty.

Our work on STM-Haskell set out to address these problems. In particular, our original paper makes the following contributions:

- We re-express the ideas of transactional memory in the setting of the purely functional language Haskell (Section 3). As we show, STM can be expressed particularly elegantly in a declarative language, and we are able to use Haskell's type system to give far stronger guaran-

tees than are conventionally possible. In particular, we guarantee “strong atomicity”¹⁵ in which transactions always appear to execute atomically, no matter what the rest of the program is doing. Furthermore transactions are compositional: small transactions can be glued together to form larger transactions.

- We present a modular form of blocking (Section 3.2). The idea is simple: a transaction calls a `retry` operation to signal that it is not yet ready to run (e.g., it is trying to take data from an empty queue). The programmer does not have to identify the condition which will enable it; this is detected automatically by the STM.
- The `retry` function allows possibly blocking transactions to be composed in *sequence*. Beyond this, we also provide `orElse`, which allows them to be composed as *alternatives*, so that the second is run if the first retries (see Section 3.4). This ability allows threads to wait for many things at once, like the Unix `select` system call—except that `orElse` composes, whereas `select` does not.

Everything we describe is fully implemented in the Glasgow Haskell Compiler (GHC), a fully fledged optimizing compiler for Concurrent Haskell; the STM enhancements were incorporated in the GHC 6.4 release in 2005. Further examples and a programmer-oriented tutorial are also available.¹⁹

Our main war cry is *compositionality*: a programmer can control atomicity and blocking behavior in a modular way that respects abstraction barriers. In contrast, lock-based approaches lead to a direct conflict between abstraction and concurrency (see Section 2). Taken together, these ideas offer a qualitative improvement in language support for modular concurrency, similar to the improvement in moving from assembly code to a high-level language. Just as with assembly code, a programmer with sufficient time and skills may obtain better performance programming directly with low-level concurrency control mechanisms rather than transactions—but for all but the most demanding applications, our higher-level STM abstractions perform quite well enough.

This paper is an abbreviated and polished version of an earlier paper with the same title.⁹ Since then there has been a tremendous amount of activity on various aspects of transactional memory, but almost all of it deals with the question of *atomic memory update*, while much less attention is paid to our central concerns of *blocking* and *synchronization* between threads, exemplified by `retry` and `orElse`. In our view this is a serious omission: locks without condition variables would be of limited use.

Transactional memory has tricky semantics, and the original paper gives a precise, formal semantics for transactions, as well as a description of our implementation. Both are omitted here due to space limitations.

2. BACKGROUND

Throughout this paper we study concurrency between threads running on a shared-memory machine; we do not consider questions of external interaction through storage systems or databases, nor do we address distributed systems. The kinds of problem we have in mind are building collection classes (queues, lists, and so on) and other data structures that concurrent threads use to maintain shared information. There are many other approaches to concurrency that we do not discuss, including data-parallel abstractions from languages like NESL² and those from the high-performance computing community such as OpenMP and MPI.

Even in this restricted setting, concurrent programming is extremely difficult. The dominant programming technique is based on *locks*, an approach that is simple and direct, but that simply does not scale with program size and complexity. To ensure *correctness*, programmers must identify which operations conflict; to ensure *liveness*, they must avoid introducing deadlock; to ensure good *performance*, they must balance the granularity at which locking is performed against the costs of fine-grain locking.

Perhaps the most fundamental objection, though, is that *lock-based programs do not compose*. For example, consider a hash table with thread-safe insert and delete operations. Now suppose that we want to delete one item A from table t_1 , and insert it into table t_2 ; but the intermediate state (in which neither table contains the item) must not be visible to other threads. Unless the implementer of the hash table anticipates this need, there is simply no way to satisfy this requirement without somehow locking out *all* other accesses to the table. One approach is to expose concurrency control methods such as `LockTable` and `UnlockTable`—but this breaks the hash table abstraction, and invites lock-induced deadlock, depending on the order in which the client takes the locks, or race conditions if the client forgets. Yet more complexity is required if the client wants to await the presence of A in t_1 —but this blocking behavior must not lock the table (else A cannot be inserted). In short, operations that are individually correct (insert, delete) cannot be composed into larger correct operations.

The same phenomenon shows up trying to compose alternative blocking operations. Suppose a procedure p_1 waits for one of two input pipes to have data, using a call to the Unix `select` procedure; and suppose another procedure p_2 does the same thing, on two other pipes. In Unix there is no way to perform a `select` between p_1 and p_2 , a fundamental loss of compositionality. Instead, Unix programmers learn awkward programming techniques to gather up all the file descriptors that must be waited for, perform a single top-level `select`, and then dispatch back to the correct handler. Again, two individually correct abstractions, p_1 and p_2 , cannot be composed into a larger one; instead, they must be ripped apart and awkwardly merged, in direct conflict with the goals of abstraction.

Rather than fixing locks, a more promising and radical alternative is to base concurrency control on *atomic memory transactions*, also known as *transactional memory*. We will show that transactional memory offers a solution to the tension between concurrency and abstraction. For ex-

ample, with memory transactions we can manipulate the hash table thus:

```
atomic {v := delete(t1, A); insert(t2, A, v)}
```

and to wait for either p_1 or p_2 we can say

```
atomic {p1 'orElse' p2}
```

These simple constructions require no knowledge of the implementation of `insert`, `delete`, p_1 or p_2 , and they continue to work correctly if these operations may block, as we shall see.

2.1. Transactional memory

The idea of transactions is not new. They have been a fundamental mechanism in database design for many years, and there has been much subsequent work on transactional memory. Larus and Rajwar provide a recent survey.¹⁴

The key idea is that a block of code, including nested calls, can be enclosed by an `atomic` block, with the guarantee that it runs atomically with respect to every other `atomic` block. Transactional memory can be implemented using *optimistic synchronization*. Instead of taking locks, an `atomic` block runs without locking, accumulating a thread-local *transaction log* that records every memory read and write it makes. When the block completes, it first *validates* its log, to check that it has seen a consistent view of memory, and then *commits* its changes to memory. If validation fails, because memory read by the method was altered by another thread during the block's execution, then the block is re-executed from scratch.

Suitably implemented transactional memory eliminates many of the low-level difficulties that plague lock-based programming. There are no lock-induced deadlocks (because there are no locks); there is no priority inversion; and there is no painful tension between granularity and concurrency. However, initial work made little progress on transactional abstractions that compose well. There are three particular problems.

Firstly, since a transaction may be rerun automatically, it is essential that it does nothing irrevocable. For example, the transaction

```
atomic {if (n > k) then launchMissiles(); S2}
```

might launch a second salvo of missiles if it were re-executed. It might also launch the missiles inadvertently if, say, the thread was de-scheduled after reading n but before reading k , and another thread modified both before the thread was resumed. This problem begs for a guarantee that the body of the `atomic` block can only perform memory operations, and hence can only make benign modifications to its own transaction log, rather than performing irrevocable input/output.

Secondly, many systems do not support synchronization *between* transactions, and those that do rely on a programmer-supplied Boolean guard on the `atomic` block.⁸ For example, a method to get an item from a buffer might be:

```
Item get () {
  atomic (n_items > 0) {... remove item ...}
}
```

The thread waits until the guard (`n_items > 0`) holds, before executing the block. But how could we take two *consecutive* items? We cannot call `get (); get ()`, because another thread might perform an intervening `get`. We could try wrapping two calls to `get` in a nested `atomic` block, but the semantics of this are unclear unless the outer block checks there are two items in the buffer. This is a disaster for abstraction, because the client (who wants to get the two items) has to know about the internal details of the implementation. If several separate abstractions are involved, matters are even worse.

Thirdly, no previous transactional memory supports *choice*, exemplified by the `select` example mentioned earlier.

We tackle all three issues by presenting transactional memory in the context of the declarative language Concurrent Haskell, which we briefly review next.

2.2. Concurrent Haskell

Concurrent Haskell²⁰ is an extension to Haskell 98, a pure, lazy, functional language. It provides explicitly forked threads, and abstractions for communicating between them. This naturally involves side effects and so, given the lazy evaluation strategy, it is necessary to be able to control exactly when they occur. The big breakthrough came from a mechanism called *monads*.²¹

Here is the key idea: a value of type `IO a` is an *I/O action* that, when performed, may do some I/O before yielding a value of type `a`. For example, the functions `putChar` and `getChar` have types:

```
putChar :: Char -> IO ()
getChar :: IO Char
```

That is, `putChar` takes a `Char` and delivers an I/O action that, when performed, prints the character on the standard output; while `getChar` is an action that, when performed, reads a character from the console and delivers it as the result of the action. A complete program must define an I/O action called `main`; executing the program means performing that action. For example:

```
main :: IO ()
main = putChar 'x'
```

I/O actions can be glued together by a *monadic bind* combinator. This is normally used through some syntactic sugar, allowing a C-like syntax. Here, for example, is a complete program that reads a character and then prints it twice:

```
main = do {c <- getChar; putChar c; putChar c }
```

As well as performing external input/output, I/O actions include operations with side effects on *mutable cells*. A value of type `IORef a` is a mutable storage cell which can hold values of type `a`, and is manipulated (only) through the following interface:

```
newIORef    :: a -> IO (IORef a)
readIORef   :: IORef a -> IO a
writeIORef  :: IORef a -> a -> IO ()
```

`newIORef` takes a value of type `a` and creates a mutable storage location holding that value. `readIORef` takes a reference to such a location and returns the value that it contains. `writeIORef` provides the corresponding update operation. Since these cells can only be created, read, and written using operations in the `IO` monad, there is a type-secure guarantee that ordinary functions are unaffected by state—for example, a pure function `sin` cannot read or write an `IORef` because `sin` has type `Float -> Float`.

Concurrent Haskell supports threads, each independently performing input/output. Threads are created using a function `forkIO`:

```
forkIO :: IO a -> IO ThreadId
```

`forkIO` takes an I/O action as its argument, spawns a fresh thread to perform that action, and immediately returns its thread identifier to the caller. For example, here is a program that forks a thread that prints 'x', while the main thread goes on to print 'y':

```
main = do {forkIO (print 'x'); print 'y' }
```

Peyton Jones provides a fuller introduction to concurrency, I/O, exceptions and cross-language interfacing (the “awkward squad” for pure, lazy, functional programming),¹⁸ and Daume III provides a general online tutorial to Haskell.⁶

3. COMPOSABLE TRANSACTIONS

We are now ready to present the key ideas of the paper. Our starting point is this: *a purely declarative language is a perfect setting for transactional memory*, for two reasons. First, the type system explicitly separates computations which may have side effects from effect-free ones. As we shall see, it is easy to refine it so that transactions can perform memory effects but not irrevocable input/output effects. Second, reads from and writes to mutable cells are explicit, and relatively rare: most computation takes place in the purely functional world. These functional computations perform many, many memory operations—allocation, update of thunks, stack operations, and so on—but none of these need to be tracked by the STM, because they are pure and never need to be rolled back. Only the relatively rare explicit operations need be logged, so a software implementation is entirely appropriate.

So our approach is to use Haskell as a kind of “laboratory” in which to study the ideas of transactional memory in a setting with a very expressive type system. As we go, we will mention primitives from the STM library, whose interface is summarized in Figure 1. In this paper, we focus on examples of how STM can be used in building simple concurrency abstractions. Our original paper⁹ formally defines the details of the design via an operational semantics which we developed alongside our implementations; we found this invaluable in highlighting interactions between the constructs—for example, what happens if an exception is raised deep inside an atomic block, nested within catch handlers and `orElse?` For the moment we return to simpler examples.

Figure 1: The STM interface.

```
-- The STM monad itself
data STM a
instance Monad STM
-- Monads support “do” notation and sequencing

-- Exceptions
throw :: Exception -> STM a
catch :: STM a -> (Exception->STM a) -> STM a

-- Running STM computations
atomic :: STM a -> IO a
retry  :: STM a
orElse :: STM a -> STM a -> STM a

-- Transactional variables
data TVar a
newTVar  :: a -> STM (TVar a)
readTVar :: TVar a -> STM a
writeTVar :: TVar a -> a -> STM ()
```

3.1. Transactional variables and atomicity

Suppose we wish to implement a resource manager, which holds an integer-valued resource. The call `(getR r n)` should acquire `n` units of resource `r`, blocking if `r` holds insufficient resource; the call `(putR r n)` should return `n` units of resource to `r`.

Here is how we might program `putR` in STM-Haskell:

```
type Resource = TVar Int
putR :: Resource -> Int -> STM ()
putR r i = do { v <- readTVar r
              ; writeTVar r (v + i) }
```

The currently available resource is held in a *transactional variable* of type `TVar Int`. The type declaration simply gives a name to this type. The function `putR` reads the value `v` of the resource from its cell, and writes back `(v + i)` into the same cell. (We discuss `getR` next, in Section 3.2.)

The `readTVar` and `writeTVar` operations both return STM actions (Figure 1), but Haskell allows us to use the same

`do {...}` syntax to compose STM actions as we did for I/O actions. These STM actions remain tentative during their execution: to expose an STM action to the rest of the system, it can be passed to a new function `atomic`, with type:

```
atomic :: STM a -> IO a
```

It takes a memory transaction, of type `STM a`, and delivers an I/O action that, when performed, runs the transaction atomically with respect to all other memory transactions. One might say:

```
main = do {...; atomic (putR r 3); ...}
```

The underlying transactional memory deals with maintaining a per-thread transaction log to record the tentative accesses made to TVars. When `atomic` is invoked, the STM checks that the logged accesses are *valid*—i.e., no concurrent transaction has committed conflicting updates to those TVars. If the log is valid then the STM *commits* it atomically to the heap, thereby exposing its effects to other transactions. Otherwise the memory transaction is rerun with a fresh log.

Splitting the world into STM actions and I/O actions provides two valuable properties, both statically checked by the type system:

- There is no way to perform general I/O within a transaction, because there is no operation that takes an IO computation and performs it in the STM monad. Hence only STM actions and pure computation can be performed inside a memory transaction. This is precisely the guarantee we sought in Section 2.1. It statically prevents the programmer from calling `launchMissiles` inside a transaction, because launching missiles is an I/O action with type `IO ()`, and cannot be composed with STM actions.
- No STM actions can be performed outside a transaction, so the programmer cannot accidentally read or write a TVar without the protection of `atomic`. Of course, one can always say `atomic (readTVar v)` to read a TVar in a trivial transaction, but the call to `atomic` cannot be omitted.

3.2. Blocking memory transactions

Any concurrency mechanism must provide a way for a thread to await an event or events caused by other threads. In lock-based programming, this is typically done using condition variables; message-based systems offer a construct to wait for messages on a number of channels; POSIX provides `select`; Win32 provides `WaitForMultipleObjects`; and STM systems to date allow the programmer to guard the atomic block with a Boolean condition (see Section 2.1).

The Haskell setting led us to a remarkably simple new mechanism for blocking. Furthermore, as we show in Sections 3.3 and 3.4, it supports composition in ways that are not possible with lock-based programming.

The idea is to provide a `retry` operation to indicate that the current atomic action is not yet ready to run to completion. Here is the code for `getR`:

```
getR :: Resource -> Int -> STM ()
getR r i = do { v <- readTVar r
              ; if (v < i) then retry
                else writeTVar r (v - i) }
```

It reads the value v of the resource and, if $v \geq i$, decreases it by i . If $v < i$, there is insufficient resource in the variable, in which case it calls `retry`. Conceptually, `retry` aborts the transaction with no effect, and restarts it at the beginning. However, there is no point in actually re-executing the transaction until *at least one of the TVars read during the attempted transaction has been written by another thread*. Happily, the transaction log (which is needed anyway) already records exactly which TVars were read. The implementation, therefore, blocks the thread until at least one of these is updated. Notice that `retry`'s type (`STM a`) allows it to be used wherever an STM action may occur.

Unlike the validation check, which is automatic and implicit, `retry` is called explicitly by the programmer. It does not indicate anything bad or unexpected; rather, it shows up when some kind of blocking would take place in other approaches to concurrency.

Notice that there is no need for the `putR` operation to remember to signal any condition variables. Simply by writing to the TVars involved, the producer will wake up the consumer. A whole class of lost-wake-up bugs is thereby eliminated.

From an efficiency point of view, it makes sense to call `retry` as early as possible, and to refrain from reading unrelated locations until after the test succeeds. Nevertheless, the programming interface is delightfully simple, and easy to reason about.

3.3. Sequential composition

By using `atomic`, the programmer identifies atomic transactions, in the classic sense that the entire set of operations that it contains appears to take place indivisibly. This is the key to sequential composition for concurrency abstractions. For example, to grab three units of one resource and seven of another, a thread can say

```
atomic (do {getR r1 3; getR r2 7})
```

The standard `do {..;..}` notation combines the STM actions from the two `getR` calls and the underlying transactional memory commits their updates as a single atomic I/O action.

The `retry` function is central to making transactions composable when they may block. The transaction above will block if either `r1` or `r2` has insufficient resource: there is no need for the caller to know how `getR` is implemented, or what condition guarantees its success. Nor is there any risk of deadlock by awaiting `r2` while holding `r1`.

This ability to compose STM actions is why we did not define `getR` as an I/O action, wrapped in a call to `atomic`.

By leaving it as an STM action, we allow the programmer to compose it with other STM actions before finally sealing it into a transaction with `atomic`. In a lock-based setting, one would worry about crucial locks being released between the two calls, and about deadlock if another thread grabbed the resources in the opposite order, but there are no such concerns here.

The STM type on an atomic action provides a strong guarantee: the *only* way the action can be executed is for it to be passed to `atomic`. *Any STM action can be robustly composed with other STM actions: the resulting sequence will still execute atomically.*

3.4. Composing alternatives

We have discussed composing transactions in *sequence*, so that both are executed. STM-Haskell also allows us to compose transactions as *alternatives*, so that only one is executed. For example, to get *either* 3 units from `r1` *or* 7 units from `r2`:

```
atomic (getR r1 3 `orElse` getR r2 7)
```

The `orElse` function is provided by the `STM` module (Figure 1); here, it is written infix, by enclosing it in backquotes, but it is a perfectly ordinary function of two arguments. The transaction `s1 `orElse` s2` first runs `s1`; if `s1` calls `retry`, then `s1` is abandoned with no effect, and `s2` is run. If `s2` also calls `retry` then the entire call retries—but it waits on the variables read by *either* of the two nested transactions (i.e., on the union of two variable sets). Again, the programmer needs know nothing about the enabling conditions of `s1` and `s2`.

Using `orElse` provides an elegant way for library implementers to defer to their caller the question of whether or not to block. For instance, it is straightforward to convert the blocking version of `getR` into one which returns a Boolean success or failure result:

```
nonBlockGetR :: Resource -> Int
              -> STM Bool
nonBlockGetR r i =
  do {getR r i ; return True}
  `orElse` return False
```

If `getR` completes normally, `nonBlockGetR` will return `True`; on the other hand, if `getR` blocks (i.e., retries), the `orElse` will try its second alternative, which succeeds immediately, returning `False`. Notice that this idiom depends on the left-biased nature of `orElse`. The same kind of construction can be also used to build a blocking operation from one that returns a Boolean result: simply invoke `retry` on receiving a `False` result:

```
blockGetR :: Resource -> Int -> STM ()
blockGetR r i =
  do {s <- nonBlockGetR r i;
      if s then return () else retry}
```

The `orElse` function obeys useful laws: it is associative and has unit `retry`:

```
M1 `orElse` (M2 `orElse` M3)
    = (M1 `orElse` M2) `orElse` M3
retry `orElse` M = M
M `orElse` retry = M
```

Haskell aficionados will recognize that STM may thus be an instance of `MonadPlus`.

3.5. Exceptions

The STM monad supports exceptions just like the IO monad, and in much the same way as (say) C#. Two new primitive functions, `catch` and `throw`, are required; their types are given in Figure 1. The question is: how should transactions and exceptions interact? For example, what should this transaction do?

```
atomic (do
  { n <- readTVar v_n
  ; lim <- readTVar v_lim
  ; writeTVar v_n (n + 1)
  ; if n > lim
  then throw
      (AssertionFailed "Urk")
  else if (n == lim) then retry
  else return ()
  ; ... write data into buffer...})
```

The programmer throws an exception if $n > \text{lim}$, in which case the `...write data...` part will clearly not take place. But what about the write to `v_n` from before the exception was thrown?

Concurrent Haskell encourages programmers to use exceptions for signalling error conditions, rather than for normal control flow. Built-in exceptions, such as divide-by-zero, also fall into this category. For consistency, then, in the above program *we do not want the programmer to have to take account of the possibility of exceptions*, when reasoning that if `v_n` is (observably) written then data is written into the buffer. We, therefore, specify that exceptions have *abort semantics*: if an atomic transaction throws an exception, then the transaction must be validated as if it had completed normally; however, no changes are committed. If validation succeeds, then the exception is propagated; but if validation fails, then the throwing of the exception may have been based on an inconsistent view of memory, so the exception is discarded and the transaction is re-executed from scratch. Abort semantics make it much easier to reason about invariants: the programmer only has to worry about the invariant being preserved when the transaction commits; exceptions raised during the transaction always restore the invariant, by definition.

Our use of exceptions to abort `atomic` blocks is a free design choice. In other languages, especially in ones where exceptions are used more frequently, it might be appropriate to distinguish exceptions that cause the enclosing `atomic`

block to abort from exceptions that allow it to commit before they are propagated.

Notice the difference between calling `throw` and calling `retry`. The former signals an error, and aborts the transaction; the latter only indicates that the transaction is not yet ready to run, and causes it to be re-executed when the situation changes.

An exception can carry a value out of the STM world. For example, consider

```
atomic (do
  { s <- readTVar svar
  ; writeTVar svar "Wuggle"
  ; is length s < 10 then
      throw (AssertionFailed s)
  else ...})
```

Here, the external world gets to see the exception value holding the string `s` that was read out of the TVar. However, since the transaction is aborted before the exception propagates, its write to `svar` is not externally observable. One might argue that it is wrong to allow even reads to “leak” from an aborted transaction, but we do not agree. The values carried by an exception can only represent a consistent view of the heap (or validation would fail, and the transaction would re-execute without propagating the exception), and it is almost impossible to debug an error condition that only says “something bad happened” while deliberately discarding all clues to what the bad thing was. The basic transactional guarantees are not threatened.

What if the exception carries a TVar *allocated* in the aborted transaction? A dangling pointer would be unpleasant. To avoid this we refine the semantics of exceptions to say that a transaction that throws an exception is aborted so far as its write effects are concerned, but its *allocation* effects are retained; after all, they are thread-local. As a result, the TVar is visible after the transaction, in the state it had when it was allocated. Cases like these are tricky, which is why we developed a full formal semantics.⁹

Concurrent Haskell also provides asynchronous exceptions which can be thrown into a thread as a signal—typical examples are error conditions like stack overflow, or when a master thread wishes to shut down a helper. If a thread is in the midst of an STM transaction, then the transaction log can be discarded without externally visible effects.

What if an exception is raised inside `orElse`? We considered a design in which, if the first alternative throws an exception, we could discard its effects and try the second alternative instead. But that would invalidate the beautiful identify which makes `retry` a unit for `orElse` and would also make `orElse` asymmetric in its treatment of exceptions (discarded from the first alternative but propagated by the second). We, therefore, chose that exceptions *do* propagate from the first alternative: the second alternative is examined only if the first one calls a `retry`.

What about catching an exception within an `atomic` block? Consider this example:


```
f :: Port Int -> STM ()
f p = do { item <- readPort p
         ; g item }
```

If `g` goes wrong (throws an exception), the author of `f` might reasonably want to ensure that the `item` is not read from the port `p` and then discarded. And indeed, if `f` is called in an atomic context, such as `atomic (f p)`, the effects of `readPort` are discarded, so that the `item` is not read. But suppose `f` is called in a context that catches the exception *before leaving the STM world*:

```
bad :: Port Int -> Port Int -> STM ()
bad p1 p2 = catch (f p1) (\exn -> f p2)
```

In our original paper we proposed that the effects of `(f p1)` would be retained and be visible to the call `(f p2)`. Furthermore, if the latter succeeds without itself throwing an exception or retrying, the effects of `(f p1)` would be permanently committed.

Ultimately we felt that this treatment of effects that precede an exception seemed inconsistent. Consider the author of `f`; in an effort to ensure that the item is indeed not read if `g` throws an exception, he might try this:

```
f :: Port Int -> STM ()
f p = do { item <- readPort p
         ; catch (g item)
              (recover exn item) }
where
  recover exn item
    = do { unReadPort p item
         ; throw exn }
```

But that relies on the existence of `unReadPort` to manually replicate the roll-back supported by the underlying STM. The conclusion is clear: the effects of the first argument of `catch` should be reverted if the computation raises an exception. Again, this works out nicely in the context of STM-Haskell because the `catch` operation used here has an STM type, which indicates to the programmer that the code is transactional.

4. APPLICATIONS AND EXAMPLES

In this section we provide some examples of how composable memory transactions can be used to build higher-level concurrency abstractions. We focus on operations that involve potentially blocking communication between threads. Previous work has shown, many times over, how standard shared-memory data structures can be developed from sequential code using transactional memory operations.^{8,11}

4.1. MVars

Prior to our STM work, Concurrent Haskell provided MVars as its primitive mechanism for allowing threads to com-

municate safely. An MVar is a mutable location like a TVar, except that it may be either *empty*, or *full* with a value. The `takeMVar` function leaves a full MVar empty, but blocks on an empty MVar. A `putMVar` on an empty MVar leaves it full, but blocks on a full MVar. So MVars are, in effect, a one-place channel.

It is easy to implement MVars on top of TVars. An MVar holding a value of type `a` can be represented by a TVar holding a value of type `Maybe a`; this is a type that is either an empty value (“Nothing”), or actually holds an `a` (e.g., “Just 42”).

```
type MVar a = TVar (Maybe a)
newEmptyMVar :: STM (MVar a)
newEmptyMVar = newTVar Nothing
```

The `takeMVar` operation reads the contents of the TVar and retries until it sees a value other than `Nothing`:

```
takeMVar :: MVar a -> STM a
takeMVar mv
  = do { v <- readTVar mv
       ; case v of
         Nothing -> retry
         Just val -> do { writeTVar mv Nothing
                        ; return val } }
```

The corresponding `putMVar` operation retries until it sees `Nothing`, at which point it updates the underlying TVar:

```
putMVar :: MVar a -> a -> STM ()
putMVar mv newval
  = do { v <- readTVar mv
       ; case v of
         Nothing -> writeTVar mv
                    (Just newval)
         Just val -> retry }
```

Notice how operations that return a Boolean success / failure result can be built directly from these blocking designs. For instance:

```
tryPutMVar :: MVar a -> a -> STM Bool
tryPutMVar mv val
  = do { putMVar mv val ; return True }
    'orElse' return False
```

4.2. Multicast channels

MVars effectively provide communication channels with a single buffered item. In this section we show how to program buffered, multi-item, multicast channels, in which items written to the channel (`writeMChan` in the interface below) are buffered internally and received once by each read-port created from the channel. The full interface is:

```

data MChan a
data port a
newMChan  :: STM (MChan a)
-- Write an item to the channel:
writeMChan :: MChan a -> a -> STM ()
-- Create a new read port:
newPort   :: MChan a -> STM (Port a)
-- Read the next buffered item:
readPort  :: Port a -> STM a

```

We represent the buffered data by a linked list, or `Chain`, of items, with a transactional variable in each tail, so that it can be extended by `writeMChan`:

```

type Chain a = TVar (Item a)
data Item a = Empty | Full a (Chain a)

```

An `MChan` is represented by a mutable pointer to the “write” end of the chain, while a `Port` points to the read end:

```

type MChan a = TVar (Chain a)
type Port a = TVar (Chain a)

```

With these definitions, the code writes itself:

```

newMChan = do {c <- newTVar Empty; newTVar c}
newPort mc = do {c <- readTVar mc; newTVar c}

readPort p
  = do { c <- readTVar p
        ; i <- readTVar c
        ; case i of
            Empty    -> retry
            Full v c' -> do {writeTVar p c';
                           return v}}

writeMChan mc v
  = do { c <- readTVar mc
        ; c' <- newTVar Empty
        ; writeTVar c (Full v c')
        ; writeTVar mc c' }

```

Notice the use of `retry` to block `readPort` when the buffer is empty. Although this implementation is very simple, it ensures that each item written into the `MChan` is delivered to every `Port`; it allows multiple writers (their writes are interleaved); it allows multiple readers on each port (data read by one is not seen by the other readers on that port); and when a port is discarded, the garbage collector recovers the buffered data.

More complicated variants are simple to program. For example, suppose we wanted to ensure that the writer could get no more than N items ahead of the most advanced reader. One way to do this would be for the writer to include a serially increasing `Int` in each `Item`, and have a shared `TVar`

holding the maximum serial number read so far by any reader. It is simple for the readers to keep this up to date, and for the writer to consult it before adding another item.

4.3. Merge

We have already stressed that transactions are *composable*. For example, to read from either of the two different multicast channels, we can say:

```
atomic (readPort p1 `orElse` readPort p2)
```

No changes need to be made to either multicast channel. If neither port has any data, the `STM` machinery will cause the thread to wait simultaneously on the `TVars` at the extremity of each channel.

Equally, the programmer can wait on a condition that involves a mixture of `MVars` and `MChans` (perhaps the multicast channel indicates ordinary data and an `MVar` is being used to signal a termination request), for instance:

```
atomic (readPort p1 `orElse` takeMVar m1)
```

This example is contrived for brevity, but it shows how operations taken from different libraries, implemented without anticipation of their being used together, can be composed. In the most general case, we can select between values received from a number of different sources. Given a list of computations of type `STM a` we can take the first value to be produced from any of them by defining a merge operator:

```
merge :: [STM a] -> STM a
merge = foldr1 orElse
```

(The function `foldr1 f` simply reduces a list $[a_1 a_2 \dots a_n]$ to the value $a_1 \text{ `f` } a_2 \text{ `f` } \dots \text{ `f` } a_n$.) This example is childishly simple in `STM-Haskell`. In contrast, a function of type

```
mergeIO :: [IO a] -> IO a
```

is unimplementable in Concurrent Haskell, or indeed in other settings with operations built from mutual exclusion locks and condition variables.

5. IMPLEMENTATION

Since our original paper there has been a lot of work on building fast implementations of `STM` along with hardware support to replace or accelerate them.¹⁴ The techniques we have used in `STM-Haskell` are broadly typical of much of this work and so we do not go into the details here. In summary, however, while a transaction is running, it builds up a private log that records the `TVars` it has accessed, the values it has read from them and (in the case of writes) the new values that it wants to store to them. When a transaction attempts to commit, it has to reconcile this log with the heap. Logically this has two steps: *validating* the transaction to check that there

have been no conflicting updates to the locations read, and then *writing-back* the updates to the TVars that have been modified.

However, the `retry` and `orElse` abstractions led us to think more carefully about how to integrate blocking operations with this general approach. Following Harris and Fraser's work⁸ we built `retry` by using a transaction's log to identify the TVars that it has read and then adding "trip wires" to those TVars before blocking: subsequent updates to any of those TVars will unblock the thread.

The `orElse` and `catch` constructs are both implemented using closed nested transactions¹⁷ so that the updates made by the enclosed work can be rolled back without discarding the outer transaction. There is one subtlety that we did not appreciate in our original paper: if the enclosed transaction is rolled back *then the log of locations it has read must be retained by the parent*. In retrospect the reason is clear—the decision of whether or not to roll back must be validated at the same atomic point as the outer transaction.

5.1. Progress

The STM implementation guarantees that one transaction can force another to abort only when the first one commits. As a result, the STM implementation is *lock-free* in the sense that it guarantees at any time that some running transaction can successfully commit. For example, no deadlock will occur if one transaction reads and writes to TVar `x` and then TVar `y`, while a second reads and writes to those TVars in the opposite order. Each transaction will observe the original value of those TVars; the first to validate will commit, and the second will abort and restart. Similarly, synchronization conflicts over TVars cannot cause cyclic restart, where two or more transactions repeatedly abort one another.

Starvation is possible, however. For example, a transaction that runs for a very long time may repeatedly be aborted by shorter transactions that conflict with it. We think that starvation is unlikely to occur in practice, but we cannot tell without further experience. A transaction may also never commit if it is waiting for a condition that never becomes true.

6. RELATED WORK

Transactions have long been used for fault tolerance in databases⁷ and distributed systems. These transactions rely on stable storage and distributed commit protocols to protect system integrity against crashes and communication failures. Transactional memory of the kind we are studying provides access to memory within a single process; it is not intended to survive crashes, so there is no need for distributed commit protocols or stable storage. It follows that many design and implementation issues are quite different from those arising in distributed or persistence-only transaction systems. TM was originally proposed as a hardware architecture^{12,24} to support nonblocking synchronization, and architectural support for this model remains the subject of ongoing research, as does the construction of efficient implementations in software. Larus and Rajwar provide a recent survey of implementation techniques.¹⁴

Transactional composition requires the ability to run transactions of arbitrary size and duration, presenting a challenge to hardware-based transactional memory designs, which are inherently resource-limited. One way for hardware to support large transactions is by virtualization,^{4,22} providing transparent overflow mechanisms. Another way is by hybrid STM designs^{5,13} that combine both hardware and software mechanisms.

After our original paper, Carlstrom et al. examined a form of `retry` that watches for updates to a specified set of locations,³ arguing that this is easier to support in hardware and may be more efficient than our form of `retry`. However, unless the watch set is defined carefully, this sacrifices the composability that `retry` provides because updates to non-watched locations may change the control flow within the transaction.

Our original paper also discusses related programming abstractions for concurrency, notably Concurrent ML's *composable events* and Scheme48's *proposals*.

7. CONCLUSION

In this paper we have introduced the ideas from STM-Haskell for composable memory transactions, providing a substrate for concurrent programming that offers far richer composition than has been available to date: two atomic actions can be glued together in sequence with the guarantee that the result will run atomically, and two atomic actions can be glued together as alternatives with the guarantee that exactly one of them will run. In subsequent work we have further enhanced the STM interface with *invariants*.¹⁰

We have used Haskell as a particularly suitable laboratory to explore these ideas and their implementation. An obvious question is this: to what extent can our results be carried back into the mainstream world of imperative programming? This is a question that we and many others have been investigating since our original paper. The ideas of composable blocking through `retry` and `orElse` seem straightforward to apply in other settings—subject, of course, to support for blocking and wake-up within the lower levels of the systems.

A more subtle question is the way in which our separation between transacted state and nontransacted state can be applied, or our separation between transacted code and nontransacted code. In Haskell, mutable state and impure code are expected to be the exception rather than the norm, and so it seems reasonable to distinguish the small amount of impure transacted code from the small amount of impure nontransacted code; both, in any case, can call into pure functions.

In contrast, in mainstream languages, most code is written in an impure style using mutable state. This creates a tension: statically separating transacted code and data retains the strong guarantees of STM-Haskell (no irrevocable calls to "launchMissiles" within a transaction, and no direct access to transacted state without going through the STM interface), but it requires source code duplication to create transacted variants of library functions and marshaling between transacted data formats and normal data formats.


Investigating the complex trade-offs in this design space is the subject of current research.^{1,16}

Whether or not one believes in transactions, it does seem likely that some combination of effect systems and/or ownership types will play an increasingly important role in concurrent programming languages, and these may contribute to the guarantees desirable for memory transactions.

Our main claim is that transactional memory qualitatively raises the level of abstraction offered to programmers. Just as high-level languages free programmers from worrying about register allocation, so transactional memory frees the programmer from concerns about locks and lock acquisition order in designing shared-memory data structures. More fundamentally, one can combine such abstractions without knowing their implementations, a property that is the key to constructing large programs.

Like high-level languages, transactional memory does not banish bugs altogether; for example, two threads can easily deadlock if each awaits some communication from the other. But the gain is very substantial: transactions provide a programming platform for concurrency that eliminates whole classes of concurrency errors, and allows the programmer to concentrate on the really interesting bits.

Acknowledgments

We would like to thank Byron Cook, Austin Donnelly, Matthew Flatt, Jim Gray, Dan Grossman, Andres Löh, Jon Howell, Jan-Willem Maessen, Jayadev Misra, Norman Ramsey, Michael Ringenburt, David Tarditi, and especially Tony Hoare, for their helpful feedback on earlier versions of this paper, and Guy Steele for his meticulous suggestions in preparing this revised version. 

References

- Abadi, M., Birrell, A., Harris, T., and Isard, M. Semantics of transactional memory and automatic mutual exclusion. *POPL'08: Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 63–74, ACM, Jan. 2008.
- Blelloch, G.E., Hardwick, J.C., Sipelstein, J., Zagha, M., and Chatterjee, S. Implementation of a portable nested data-parallel language. *J. Parallel Distrib. Comput.*, 21 (1): 4–14, 1994.
- Carlstrom, B.D., McDonald, A., Chafi, H., Chung, J., Minh, C.C., Kozyrakis, C., and Olukotun, K. The Atomos transactional programming language. *PLDI'06: Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 1–13, ACM, June 2006.
- Chung, J., Minh, C.C., McDonald, A., Skare, T., Chafi, H., Carlstrom, B.D., Kozyrakis, C., and Olukotun, K. Tradeoffs in transactional memory virtualization. *ASPLOS'06: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 371–381, ACM, Oct. 2006.
- Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., and Nussbaum, D. Hybrid transactional memory. *ASPLOS'06: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 336–346, ACM, Oct. 2006.
- Daume III, H. Yet another Haskell tutorial. <http://www.cs.utah.edu/~hal/docs/daume02yaht.pdf>, 2006.
- Gray, J., and Reuter, A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., 1992.
- Harris, T., and Fraser, K. Language support for lightweight transactions. *OOPSLA'03: Proceedings of the 18th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 388–402, ACM, Oct. 2003.
- Harris, T., Marlow, S., Peyton Jones, S., and Herlihy, M. Composable memory transactions. *PPoPP'05: Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 48–60, ACM, June 2005.
- Harris, T., and Peyton Jones, S. Transactional memory with data invariants. *TRANSACT'06: Proceedings of the 1st ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing*, June 2006.
- Herlihy, M., Luchangco, V., Moir, M., and Scherer, III, W.N. Software transactional memory for dynamic-sized data structures. *PODC'03: Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing*, pp. 92–101, ACM, July 2003.
- Herlihy, M. and Moss, J.E.B. Transactional memory: Architectural support for lock-free data structures. *ISCA'93: Proceedings of the 20th International Symposium on Computer Architecture*, pp. 289–300, ACM, May 1993.
- Kumar, S., Chu, M., J. Hughes, C., Kundu, P., and Nguyen, A. Hybrid transactional memory. *PPoPP'06: Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 209–220, ACM, Mar 2006.
- Larus, J., and Rajwar, R. *Transactional Memory (Synthesis Lectures on Computer Architecture)*. Morgan & Claypool Publishers, 2007.
- Martin, M., Blundell, C., and Lewis, E. Subtleties of transactional memory atomicity semantics. *IEEE Comput. Archit. Lett.* 5 (2):17, 2006.
- Moore, K.F. and Grossman, D. High-level small-step operational semantics for transactions. *POPL'08: Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 51–62, ACM, Jan. 2008.
- Moss, E.B. Nested transactions: An approach to reliable distributed computing. Tech. Rep. MIT/LCS/TR-260, Massachusetts Institute of Technology, Apr. 1981.
- Peyton Jones, S. Tackling the awkward squad: Monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell. *Engineering Theories of Software Construction, Marktoberdorf Summer School 2000*.
- Peyton Jones, S. Beautiful concurrency. In *Beautiful Code* (2007), A. Oram and G. Wilson, Eds., O'Reilly.
- Peyton Jones, S., Gordon, A., and Finne, S. Concurrent Haskell. *POPL'96: Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 295–308, ACM, Jan. 1996.
- Peyton Jones, S. and Wadler, P. Imperative functional programming. *POPL'93: Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 71–84, ACM, Jan. 1993.
- Rajwar, R., Herlihy, M., and Lai, K. Virtualizing transactional memory. *ISCA'05: Proceedings of the 32nd International Symposium on Computer Architecture*, pp. 494–505, IEEE Computer Society, June 2005.
- Shavit, N., and Touitou, D. Software transactional memory. *PODC'95: Proceedings of the 14th ACM Symposium on Principles of Distributed Computing*, pp. 204–213, ACM, Aug. 1995.
- Stone, J.M., Stone, H.S., Heidelberger, P., and Turek, J. Multiple reservations and the Oklahoma update. *IEEE Parallel and Distributed Technology* 1(4):58–71, 1993.
- Sutter, H. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobbs's J.* (March 2005).

Tim Harris (tharris@microsoft.com)
Microsoft Research

Simon Marlow (simonmar@microsoft.com)
Microsoft Research

Simon Peyton Jones (simonpj@microsoft.com)
Microsoft Research

Maurice Herlihy (mph@cs.brown.edu)
Brown University

© 2008 ACM 0001-0782/08/0800 \$5.00

Epic Systems Corporation **Technical Services Engineer**

Epic's Technical Services team is responsible for our clients' happiness after the systems are installed. They create valuable relationships by listening well to customer concerns and championing clients' needs. They work with IT staff at customer sites to quickly resolve technical issues and perform necessary programming, helping to ensure that every customer gets the most out of an Epic software investment.

Candidates should have a bachelor's degree (all majors considered), a history of academic success demonstrated by a minimum 3.2 cumulative GPA, strong analytical and reasoning skills, and be eligible to work in the U.S. without sponsorship. Because we train internally, no prior technical experience is necessary, but exposure to programming is a plus. Relocation to Madison, Wisconsin is required and reimbursed.

Epic Systems Corporation **Software Engineer / Developer**

Our small teams of software engineers participate in all aspects of the development process, from meeting customers to system design through quality assurance and delivery. Their goal is to create easy-to-use systems with optimal workflows that manage large amounts of data with sub-second response times and rock-solid stability. Our continued success in these areas is shown by Epic software systems' top-rated industry reviews. New functionality and systems are being developed daily that extend current capabilities and break new ground in the industry.

Candidates should have a bachelor's degree in Computer Science, Math, Electrical Engineering or Computer Engineering and a history of academic success demonstrated by a minimum 3.2 cumulative GPA. Relocation to Madison, Wisconsin is required and reimbursed. Visa sponsorship is available.

Health Research, Inc. **Supervisor of Data Processing**

Health Research, Inc. (HRI) seeks a Supervisor of Data Processing, in the Bureau of Computer Systems Development, to enhance and maintain its web based application, Statewide Perinatal Data System (SPDS), in Albany, NY.

SPDS provides a mechanism for issuing birth certificates, & collecting birth data from hospitals & birthing centers in New York State, exclusive of New York City. The incumbent will be required to work w/ SPDS customers to define business needs, determine scope of projects, set priorities, assess the impact of potential chang-

es, & develop project plans. The incumbent will oversee & assist subordinate staff in the development of use case modes & change specifications, identification of schema changes, development & modification of application (Java, JSP) code, creation of test plans, performance of testing activities, & implementation of changes & enhancements. As part of these responsibilities the incumbent will be required to coordinate & oversee the activities of subordinate staff working on SPDS. The Supervisor of Data Processing is required to write, modify, & test application (Java, JSP) code. Other related duties as assigned.

Min Quals: Master's degree in Computer Science or Engineering & 3 yrs of professional computer programming & systems analysis experience, 1 yr must have included supervisory experience or in a project leadership role. 3 yrs of professional exp must include: development & maintenance of Enterprise Java (J2EE) Web applications (JDBC, Java Servlets, Java Server Pages, JavaScript, & Enterprise Java Beans (EJB)); & BEA Weblogic Application Server (Version 8 or higher). 2 yrs exp w/ Oracle (8i or higher) SQL databases, defining business rules & developing case models using the Rational toolset. 1 yr exp w/ Itext, FileNet API for Java, and UNIX.

To apply, please visit HRI's web site at: <http://www.healthresearch.org/jobs>.

No phone calls or faxes accepted
AA/EOE/M/W/D/V

Fortify **Software Security Consultant**

Deliver on-site software security professional services for USAF Application Software Assurance Center of Excellence

- Install, customize, and support Fortify products
- Train, mentor customers
- Conduct source code analysis, application security testing; audit results
- Develop vulnerability remediation plans, reports
- Travel to military installations worldwide
- Must be able to obtain security clearance

Lime Spot LLC/LimeBits **Software build + release engineer/developer,** **LimeBits**

LimeBits is developing an innovative software platform intended to empower people to share their own creations and to collaborate on group creations. You'll join the LimeBits team and help overturn the ancient traditions of software development. Your well-organized build/test/release system will help our team deploy robust, fresh software rapidly and frequently. As a key member of the team, you have a blue-sky opportunity to lead us in

- ▶ automating, streamlining, coordinating, optimizing, and documenting our build/test/release processes,
- ▶ managing our source code, common library development, configuration, team wiki, bug tracking,
- ▶ managing branch/merge and database upgrade migration,
- ▶ developing a bare-metal install/recovery procedure,
- ▶ implementing monitors, reports, and statistics for build/test/release,
- ▶ designing, developing, running, and analyzing program/system tests for correctness and performance,
- ▶ designing and developing software engineering team tools,
- ▶ developing additional features of the LimeBits platform,
- ▶ driving best practices for build/test/release.

QUALIFICATIONS:

- ▶ Strong experience specialized in build/test/release engineering.
- ▶ Experience with web service development, build, test, and release.
- ▶ Solid knowledge of all phases of software development and deployment.
- ▶ Expert knowledge of source code control, preferably including Subversion and Git.
- ▶ Programming experience with C, JavaScript, Postgres, Apache, XML.
- ▶ Proficiency in build scripting with Make and Python, as well as Ruby, Perl, or shell.
- ▶ Experience with git or Mercurial.
- ▶ Strong experience developing software test scripts, including Selenium.
- ▶ Strong experience with Linux/Unix and open-source software.
- ▶ Ability to work with frequent software builds and releases.
- ▶ Interest in open-source software and content sharing.
- ▶ Self-motivated, critical thinker with strong technical background. Curiosity, imagination, intuition. Excellent communication, organization, and teamwork skills.
- ▶ Bachelor's, Master's, or Doctorate in Computer Science, Software Engineering, or related field.
- ▶ Bonus: Software configuration and packaging for Windows.
- ▶ Bonus: Knowledge of Rake, WebDAV, XSLT, MySQL, functional programming or server-side JavaScript frameworks.

COMPANY:

LimeBits is a startup project in the Lime Group, home to such companies as Lime Wire, Lime Brokerage, and Tower Research. The Lime Group companies offer a dynamic and intellectually stimulating work environment. While we work hard, we also play hard and believe in supporting our team. We provide free lunches, snacks, and beverages, tickets to NY events, 5 weeks vacation, and great views from our garden roof deck.



Windows Kernel Source and Curriculum Materials for Academic Teaching and Research.

The Windows® Academic Program from Microsoft® provides the materials you need to integrate Windows kernel technology into the teaching and research of operating systems.

The program includes:

- **Windows Research Kernel (WRK):** Sources to build and experiment with a fully-functional version of the Windows kernel for x86 and x64 platforms, as well as the original design documents for Windows NT.
- **Curriculum Resource Kit (CRK):** PowerPoint® slides presenting the details of the design and implementation of the Windows kernel, following the ACM/IEEE-CS OS Body of Knowledge, and including labs, exercises, quiz questions, and links to the relevant sources.
- **ProjectOZ:** An OS project environment based on the SPACE kernel-less OS project at UC Santa Barbara, allowing students to develop OS kernel projects in user-mode.

These materials are available at no cost, but only for non-commercial use by universities.

For more information, visit www.microsoft.com/WindowsAcademic or e-mail compsci@microsoft.com.

Wisconsin Alumni Research Foundation Postdoctoral Positions in Computing and Informatics - Biology & Medicine UW-Madison / Morgridge Institute for Research

The University of Wisconsin-Madison, with support from the Morgridge Institute for Research, has several postdoctoral research positions in computation and informatics for researchers wishing to solve biomedical problems requiring strengths in both computational and biological sciences.

These positions are being offered in cooperation with the Computation and Informatics in Biology and Medicine Training Program (CIBM; www.cibm.wisc.edu). The 45 CIBM faculty span 15 different departments and five colleges at UW-Madison and includes several faculty at the Marshfield Clinic Research Foundation (located about 100 miles north of Madison). These positions are open to both US and non-US Citizens with a Ph.D., or equivalent, in computer science. The positions are funded for up to two years, renewable for a second year pending satisfactory progress, with an annual stipend up to \$65,000 per year.

The research focus is in the development of:

- ▶ Novel bioinformatics algorithms to analyze molecular data, including genome sequences, proteins (levels, interactions, structures), and regulatory pathways,
- ▶ New tools for imaging and genetic analysis,
- ▶ Development of health delivery systems,
- ▶ Translational bench-to-bedside medicine

For more information about the position and application materials, please contact Louise Pape

KUWAIT UNIVERSITY FACULTY OF SCIENCE Kuwait

The Department of **Mathematics and Computer Science** in the Faculty of Science at Kuwait University invites applications for appointment of faculty members starting from September 2008, for the academic year 2008/2009, in one of the following areas:

Networks, Operating Systems, Mobile Computing, Multimedia Systems, Computer Architecture, Theoretical Computer Science and Parallel & Distributed Computing

Required Qualifications:

- Ph.D. degree in the area of specialization from a reputable University.
- The applicants GPA in first university degree should be 3 points out of 4 (or equivalent).
- Research experience and significant publications in refereed international journals.
- Full command of teaching in English.
- Minimum of 5 years in University teaching experience in the specified field.

The successful candidates are expected to have a strong commitment and dedication to quality teaching and research.

Benefits include attractive tax-free salary according to rank and teaching experience (Professor's monthly salary varies from 2950 to 3192 KD., Associate Prof.'s salary varies from KD. 2265 to 2507, Assistant Professor's monthly salary varies from KD. 1830 to 2070 - [KD.1 = \$3.40]), annual air tickets for the faculty member and his/her family (spouse and up to three children under the age of 20), a one time settling-in allowance, housing allowance, free national health medical care, paid mid-term holidays and summer vacations, and end-of-contract gratuity. The University also offers an excellent academic environment and financial support for research projects.

To apply, send by express mail/courier service or email, within two weeks of the date of announcement, a completed application form, updated curriculum vitae (including mailing address, phone and fax numbers, e-mail address, academic qualifications, teaching and research experience, and a list of publications in professional journals up to 10 reprints), three copies of Ph.D., Masters, and Bachelor certificates and transcripts (An English translation of all documents in other languages should be enclosed), a copy of the passport, three recommendation letters, and names and addresses of three persons well-acquainted with the academic and professional work of the applicant. Please use PDF format for all electronic application materials. Applications and inquiries should be addressed to:

Dr. Salem Al-Yakoob
Chairman

Department of Mathematics and Computer Science
Faculty of Science, Kuwait University
P.O. Box 5969, Safat, 13060, Kuwait
Tel: (965) 4813129
Fax: +965 4817201
E-Mail: math@sci.kuniv.edu.kw
<http://www.sci.kuniv.edu.kw>

at lpape@wisc.edu or call 608-265-7935.

For more information about the Morgridge Institute for Research please visit www.morgridgeinstitute.org or contact Laura M. Heisler, Ph.D., Program Developer, Morgridge Institute for Research, 614 Walnut St., Madison, WI 53726 608.261.1022, heisler@morgridgeinstitute.org

The National Academies Program Officer

The Program Officer for the Computer Science and Telecommunications Board will be responsible for developing and managing two or more of the organization's projects with minimal oversight by the board director/senior program officers. Develop program/project strategy and budget, direct the work of other staff as appropriate, and ensure projects meet objectives. Develop prospectuses, assemble committees/panels and serve as staff liaison between committee/panel members and the national academies. Organize and assist committees, including planning meetings, performing literature searches, and maintaining websites.

A pioneer in framing and analyzing Internet policy, the Computer Science and Telecommunications Board (CSTB) provides independent assessments of technical and public policy issues relating to computing and communications. Composed of leaders in information technology and complementary fields from industry and academia, CSTB is unique in its scope and its interdisciplinary approach to technical, economic, social, and policy issues.



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to jonathan.just@acm.org. Please include text, and indicate the issue/ or issues where the ad will appear, and a contact name and number.

Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. **NO PROOFS** can be sent. Classified line ads are **NOT** commissionable.

Rates: \$295.00 for six lines of text, 40 characters per line. \$80.00 for each additional three lines. The **MINIMUM** is six lines.

Deadlines: Five weeks prior to the publication date of the issue (which is the first of every month). Latest deadlines: <http://www.acm.org/publications>

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at:

<http://campus.acm.org/careercenter>
Ads are listed for a period of six weeks.

For More Information Contact:

JONATHAN JUST
Director of Media Sales
at 212-626-0687 or
jonathan.just@acm.org

Calendar of Events

August

August 24-27

The 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV,
Contact: Ying Li,
Phone: 425-703-8739,
Email: yingli@microsoft.com

September

September 1-3

8th International Conference on Intelligent Virtual Agents, Tokyo, Japan,
Contact: Helmut Prendinger,
Email: helmut@nii.ac.jp

September 2-5

10th International Conference on Human Computer Interaction with Mobile Devices and Services,
Contact: Henri Hofte,
Phone: 31-575-516319,
Email: henri.terhoft@telin.nl

September 8-11

Principles and Practice of Programming in Java 2008, Modena, Italy,
Contact: Giacomo Cabri,
Phone: 39-059-2056190,
Email: giacomo.cabri@unimore.it

September 8-12

12th International Software Product Line Conference 2008, Limerick, Ireland,
Contact: Lero Klaus Pohl,
Email: klaus.pohl@sse.uni-due.de

September 15-19

ASE '08: International Conference on Automated Software Engineering, L'Aquila, Italy,
Contact: Paola Inverardi,
Phone: 39-862-433-127,
Email: inverard@di.univaq.it

September 16-19

ACM Symposium on Document Engineering, Brazil,
Contact: Maria da Graca Campos Pimentel,
Phone: 55-16-3373-9657,
Email: mgp@icmc.usp.br

September 16-19

ECCE08: European Conference on Cognitive Ergonomics, Madeira, Portugal,
Contact: Joaquim A. Jorge,
Phone: 351-21-3100363,
Email: jaj@inesc.pt

September 20-23

The 10th International Conference on Ubiquitous Computing, Seoul, South Korea,
Contact: Joseph McCarthy,
Phone: 650-804-6987,
Email: joe@interrelativity.com

September 22-23

Multimedia and Security Workshop, Oxford, United Kingdom,
Sponsored: SIGMM,
Contact: Andrew David Ker,
Phone: +44 1865 276602,
Email: adk@comblab.ox.ac.uk

September 28-October 2

ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (formerly UML), Toulouse, France,
Sponsored: SIGSOFT,
Contact: Jean-Michel Bruel,
Phone: +33 686 002 902,
Email: bruel@univ-pau.fr

September 29-30

3rd International Conference on the Pragmatic Web, Uppsala, Sweden,
Contact: Par J. Agerfalk,
Phone: 46-18-4711064,
Email: par.agerfalk@dis.uu.se

October

October 1-31

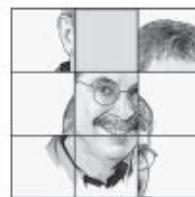
RecSys '08: ACM Conference on Recommender Systems, Lausanne, Switzerland,
Contact: Pearl Pu, Phone: 0041-216936081,
Email: pearl.pu@epfl.ch

October 1-3

International Conference on Human-Computer Interaction in Aeronautics, Toulouse, France,
Contact: Guy A. Boy,
Phone: 336-633-3682,
Email: guy.boy@eurisco.org

October 6-7

Eighth Workshop on Hot Topics in Networks, Alberta, Canada,
Contact: Carey L. Williamson,
Phone: 403-220-6780,
Email: carey@cpsc.ucalgary.ca



DOI:10.1145/1378704.1378726

Peter Winkler

Puzzled Delightful Graph Theory

Welcome to the new puzzle column. Each column will present three puzzles. The first two will have known (and usually elegant) solutions that will appear in the next issue of *Communications*. The third will be an open problem; good luck with that one.

Readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

We start with three delightful graph-theoretic puzzles. Here we go.

1. Since this is an election year, it seems appropriate to visit that impressionable town in which, every evening, each citizen calls all his (or her) friends (always an odd number) and re-chooses his party affiliation—Republican or Democrat—the next day, in accordance with the majority of his friends at the time of the call. Can you show that, after a while, party affiliations will be the same on every alternate day?

2. The island of Foosgangland boasts a complex web of footpaths. Each section of path, from one intersection to the next, is identified by a different number. If you happen to take a walk in Foosgangland, the “length” of your walk is the number of path sections you traverse, and your walk is “increasing” if the path numbers you encounter are always go up. Prove that there is someplace on the island where you can take an increasing walk whose length is at least the average number of paths meeting at the intersections in Foosgangland.

3. In a graph-coloring game, a finite graph G and a palette of k colors are fixed. Alice and Bob alternately choose an uncolored vertex of G and color it with a color not previously used on any neighboring vertex. Alice, who goes first, wins if all the vertices get colored; but if anyone gets stuck before that happens, Bob wins. (The game is described in an article—Bartnicki, T., Grytczuk, J., Kierstead, H.A., and Zhu, X. The map-coloring game. *American Mathematical Monthly* 114, 9 (Nov. 2007), 793–803—that pointed out that the following question remains embarrassingly open: Are there a G and a k such that Alice wins on G with k colors, but Bob wins with $k+1$?)

Peter Winkler (puzzled@cacm.acm.org) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH. He has written two puzzle books: *Mathematical Puzzles: A Connoisseur's Collection* and *Mathematical Mind-benders*, both published by A K Peters, Ltd., Wellesley, MA.

ICL 2008 Special Track Call for Papers

The 11th International Conference
"Interactive Computer aided Learning"
ICL2008 from September 24-26, 2008 in
Villach, Austria has again a Special Track

School and IT

International Conference ICL

This Special Track will be organized in cooperation with:

- Federal Ministry for Education, the Arts and Culture,
- Austrian Computer Society (OCG),
- European Distance Education Network (EDEN)

Conference chair

M. Auer (Carinthia Tech
Institute Villach)

Special track chair

Linmi Tao (tao.linmi@gmail.com)
Tsinghua University Beijing, China

Proceedings

The proceedings will be published
on CD in cooperation with the Kassel
University Press (own ISBN number).

Submission of papers

Extended abstracts should be
submitted using the **Electronic
Submission System**.

The extended abstract should comprise
up to two pages, informing the
program committee about the aim of
the approach (study, tool) reported,
experiences gained and the form and
result of evaluations conducted.

Proposals for tutorials and the exhibition
also may be submitted in a short form to:
info@icl-conference.org .

More information

<http://www.icl-conference.org> info@
icl-conference.org

General information

The conference will be organized by
the Carinthia Tech Institute in Villach.
Conference venue is the Conference
Center Villach.

Topics of interest

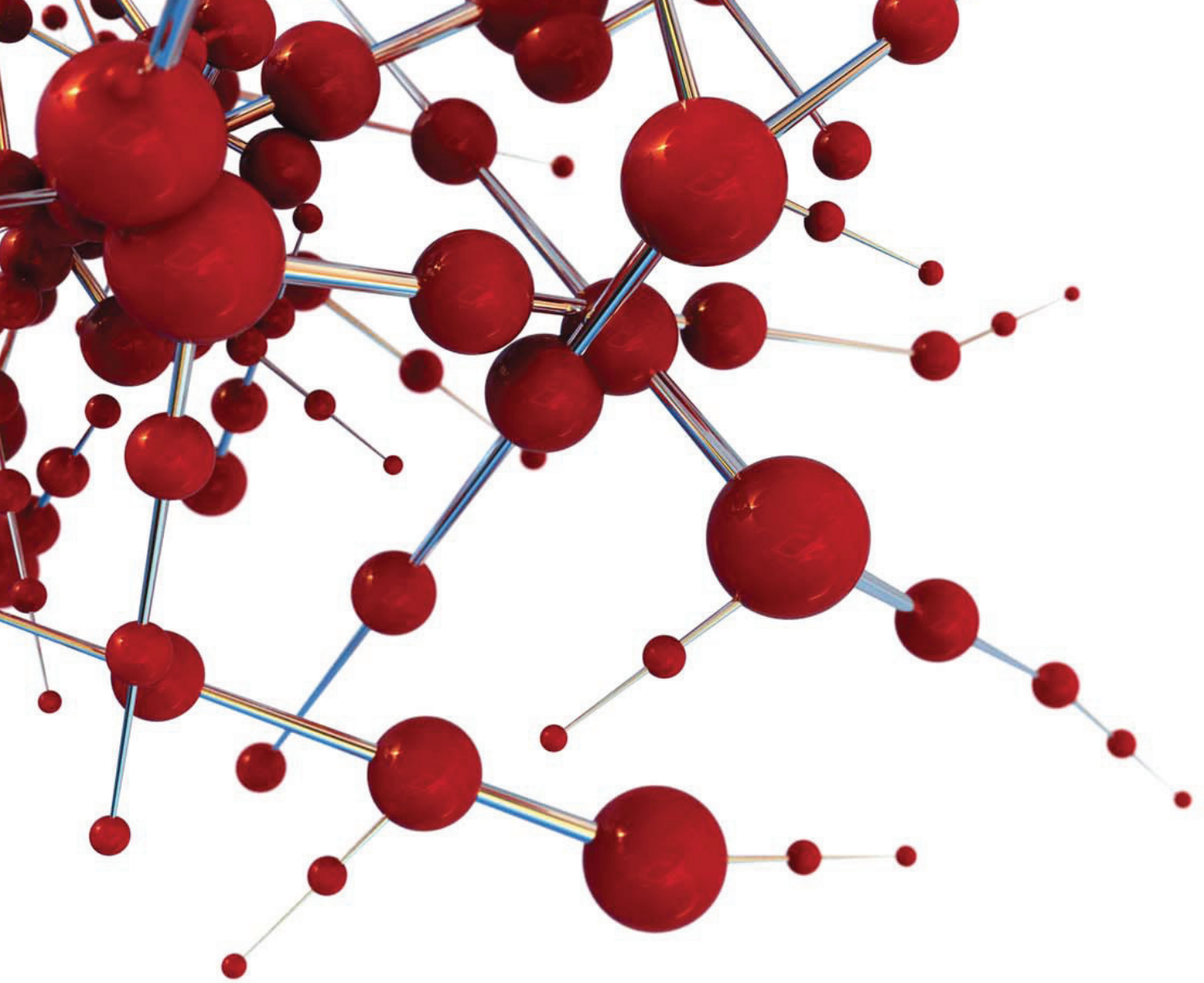
- IT & School development
- Didactical approaches
- Experiences from pilot projects
- Content development and management
- Education and further education in IT
- Multicultural and transnational approaches

Types of contributions

- Full Papers: 20 minutes presentation followed by
a panel discussion
- Interactive Demonstrations: 15 minutes demonstration.
- Poster Presentations

Other opportunities to participate:

- Exhibit at the ICL products and developments of e-learning.



**CONNECT WITH OUR
COMMUNITY OF EXPERTS.**

www.reviews.com



Association for
Computing Machinery

Reviews.com

They'll help you find the best new books
and articles in computing.

Computing Reviews is a collaboration between the ACM and Reviews.com.