COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

03/2009 VOL.52 NO.03

Being Human in the Digital Age

Better Scripts, Better Games

The Evolution of Virtualization

ACM Fellows

An Interview with C.A.R. Hoare

Association for Computing Machinery

CHI 2009

DIGITAL LIFE NEW WORLD

Join us in Boston, MA for the 27th Annual CHI Conference, the premier international forum for all aspects of human-computer interaction. Computing is reaching into all parts of modern life. CHI 2009 brings together people working on the design, evaluation, implementation, and study of interactive computing systems for human use. CHI serves as a forum for the exchange of ideas among computer scientists, human factors scientists, psychologists, social scientists, system designers, usability professionals, and end users.

www.chi2009.org

acm Association for Computing Machinery SIGCHI

Congratulations
2008 ACM Fellows

ACM honors 44 new inductees as ACM Fellows in recognition of their extraordinary leadership and ongoing contributions to the development of the Information Age

Martín Abadi
Microsoft Research Silicon Valley / University of California, Santa Cruz

Gregory Abowd
Georgia Institute of Technology

Alexander Aiken
Stanford University

Sanjeev Arora
Princeton University

Hari Balakrishnan
Massachusetts Institute of Technology

William Buxton
Microsoft Research

Kenneth L. Clarkson
IBM Almaden Research Center

Jason (Jingsheng) Cong
University of California at Los Angeles

Perry R. Cook
Princeton University

Stephen A. Cook
University of Toronto

Jack W. Davidson
University of Virginia

Umeshwar Dayal
Hewlett-Packard Laboratories

Xiaotie Deng
City University of Hong Kong

Jose J. Garcia-Luna-Aceves
University of California, Santa Cruz / Palo Alto Research Center

Michel X. Goemans
Massachusetts Institute of Technology

Patrick Hanrahan
Stanford University

Charles H. House
Stanford University MediaX Program

Watts S. Humphrey
SEI, Carnegie Mellon University

Alan C. Kay
Viewpoints Research Institute

Joseph A. Konstan
University of Minnesota

Roy Levin
Microsoft Research Silicon Valley

P. Geoffrey Lowney
Intel Corporation

Jitendra Malik
University of California, Berkeley

Kathryn S. McKinley
The University of Texas at Austin

Bertrand Meyer
ETH Zurich

John C. Mitchell
Stanford University

Joel Moses
Massachusetts Institute of Technology

J. Ian Munro
University of Waterloo

Judith S. Olson
University of California at Irvine

Lawrence C. Paulson
University of Cambridge Computer Laboratory

Hamid Pirahesh
IBM Almaden Research Center

Brian Randell
Newcastle University

Michael K. Reiter
University of North Carolina at Chapel Hill

Jennifer Rexford
Princeton University

Jonathan S. Rose
University of Toronto

Mendel Rosenblum
Stanford University

Rob A. Rutenbar
Carnegie Mellon University

Tuomas Sandholm
Carnegie Mellon University

Vivek Sarkar
Rice University

Mark S. Squillante
IBM Thomas J. Watson Research Center

Per Stenström
Chalmers University of Technology

Madhu Sudan
Massachusetts Institute of Technology

Richard Szeliski
Microsoft Research

Douglas Terry
Microsoft Research Silicon Valley

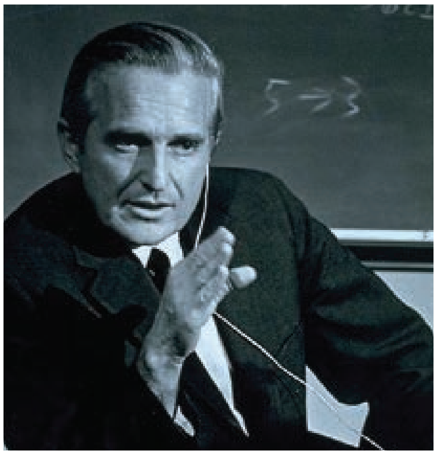
http://fellows.acm.org

Association for Computing Machinery

Advancing Computing as a Science & Profession

COMMUNICATIONS OF THE ACM

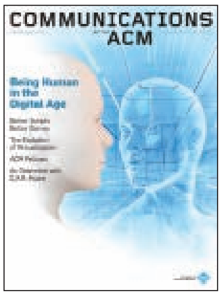
Association for Computing Machinery
Advancing Computing as a Science & Profession


Douglas C. Engelbart

PHOTOGRAPHS COURTESY OF BOOTSTRAP ALLIANCE

03/2009
VOL. 52 NO. 3

ILLUSTRATION BY ANDY GILMORE



About the Cover:
The relationship between humans and computers has changed radically in the last quarter century—along with the sociotechnical landscape—calling the relevancy of current HCI methods into question. Illustration by Bryan Christie Design.

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.

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

CONTACT POINTS

Copyright permission
permissions@cacm.acm.org
Calendar items
calendar@cacm.acm.org
Change of address
acmcoa@cacm.acm.org
Letters to the Editor
letters@cacm.acm.org

WEB SITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

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

Media Kit acmmediasales@acm.org

BPA Audit Pending

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://www.acm-media.org 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.

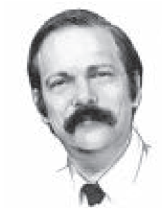
POSTMASTER

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.

PLEASE RECYCLE THIS MAGAZINE



DOI:10.1145/1467247.1467248

Moshe Y. Vardi

"Yes, It Can Be Done"

The 2008 presidential campaign slogan "Yes, We Can" is the English translation of the United Farm Workers' 1972 slogan "Sí, se puede," or "Yes, it can be done."

In 2005, I had a conversation with a member of ACM's Publications Board about the (then nascent) idea of revitalizing *Communications*. I was very pessimistic then, saying, "It cannot be done." About a year later, in the fall of 2006, I undertook that very task. Now, it is March of 2009, and we can say, "Yes, it can be done."

Why was I wrong in 2005? To start, I underestimated the determination of ACM's leadership to turn *Communications* around. I also underestimated the willingness of *Communications*' staff to undertake a radical change in the way they go about their jobs. Most of all, I underestimated ACM membership's intense desire for change and willingness to volunteer their effort toward the development of a flagship publication of which we can all be proud.

In my January 2008 editorial, I described *Communications'* editorial model as we envisioned it. Since I view this publication as a joint project between our Editorial Board and ACM's membership, it is important, I believe, that our editorial model be well understood. In January, I explained how our News and Viewpoints boards operate.

Our Practice Board, chaired by Stephen Bourne, with James Maurer as publisher, has a dual personality. On one hand, it is part of *Communications*' Editorial Board, with responsibility for developing the content for the Practice section. On the other hand, that same board is also *Queue*'s Editorial Board, with the closely related, but independent, task of developing practitioner content for ACM, primarily through the *Queue* Portal, at queue.acm.org. This board thrives on intense face-to-face interaction, meeting monthly to discuss emerging technologies. They identify topics of current interest to software architects, project leaders, IT managers, and corporate decision makers. The board also identifies potential authors and then commissions them to develop articles, under the guidance of board members and invited guest experts.

The Contributed Articles Board, chaired by Al Aho and Georg Gottlob, operates like a traditional editorial board of a scientific journal. Unsolicited manuscripts are submitted via Manuscript Central, a Web-based system for facilitating a fully online review process. As this board handles both Contributed and Review articles, the co-chairs assign each submission to an associate editor, who oversees a scholarly review process. The co-chairs and associate editors can decide to decline a paper without further review, if they judge it does not fit our new content model.

The bar for acceptance is very high; articles must be of the highest quality and reach out to a very broad technical audience. A significant fraction of the submissions fit *Communications*' previous editorial model and must be declined. A major task of this board is to encourage submissions by authors inspired by the new editorial model. It is fair to say that attracting high-quality Contributed and Review articles is an ongoing effort.

The Research Highlights Board aims to leverage the unique feature of computing research from our highly selective conferences. Their goal is to provide readers with a collection of outstanding research articles, selected from the broad spectrum of computing-research conferences, and reposition them for a far more diversified audience. Submissions are first nominated by Board Members or Approved Nominating Organizations and are subject to final selection by the Board. Authors are invited to rewrite and expand the scope of their research papers to address *Communications*' broad readership.

Each of these articles is accompanied by a Technical Perspective essay, providing readers with a one-page overview of the underlying motivation and important ideas of the featured research as well as its scientific and practical significance. Technical Perspective essays are written by rising stars and established luminaries invited by the Board. The challenge for this Board is to develop a reach into hundreds of computing-research conferences. So far, only about 10 ACM SIGs have applied to become Approved Nominating Organizations. We hope to see more SIGs applying this year, as well as non-ACM organizations.

This, in a nutshell, is how *Communications*' editorial work is carried out. I've also tried to give you a sense of the ongoing challenges. Producing a top-notch flagship publication is an evolving project. I am pleased with the progress we have made so far, and am acutely aware of the efforts required to sustain and improve upon the quality of this magazine. Yes, it can be done, if we, ACM members, collectively shoulder this effort.

Moshe Y. Vardi, EDITOR-IN-CHIEF

Emerging Software Technologies

DISNEY'S CONTEMPORARY RESORT, ORLANDO, FL
October 25-29 2009

OOPSLA

O R L A N D O **2 0 0 9**

Scaling: Multi-core to Cloud
Mashups of Models, Data and Code
Tools for Reliability and Evolution
Enterprise Agile Management

Conference Call for Submissions

March 19, 2009
Due date for Research Program, Onward!, Practitioner report, Educators' Symposium, Essays, and proposals for Tutorials, Panels, Workshops, and DesignFest®

July 2, 2009
Due date for Posters, Demonstrations, Doctoral Symposium, Onward! Films, Student Research Competition, and Student Volunteers

PROGRAM CHAIR
Gary T. Leavens
University of Central Florida
papers@oopsla.org

CONFERENCE CHAIR
Shail Arora, Gradepoint Inc.
chair@oopsla.org

ONWARD! CHAIR
Bernd Brügge
Technische Universität München
chair@onward-conference.org

Association for Computing Machinery
Advancing Computing as a Science & Profession

OOPSLA is sponsored by ACM SIGPLAN in cooperation with SIGSOFT

www.oopsla.org/submit

DOI: 10.1145/1467247.1467249 Scott E. Delman

Communications' Web Site to Launch in March

“The site will extend beyond *Communications*' current reach and help bring us closer to fulfilling the flagship's original promise as the primary “communication” tool in the field of computing.”

2008 was a year of significant change for *Communications*. The same will be the case in 2009. After a successful relaunch of the print magazine last year, ACM is getting ready to launch a new *Communications'* Web site, which will go live this month. The new site will complement the magazine by providing an easy access point to all the content found in the magazine's print pages, but perhaps more importantly the site will extend beyond *Communications'* current reach and help bring us closer to fulfilling the flagship's original promise as the primary “communication” tool in the field of computing.

Let me say a few words about the new site. Many in the community are now used to downloading *Communications'* articles from the ACM Digital Library, reading the print publication on the train or plane, or scanning through the pages of the Digital Edition on your desktop or mobile device (as an aside, the iPhone version is worth trying). For those of you who have your preferred way of digesting and archiving the articles published each month, nothing should change and we will do our best to continue to improve the experience for you. The new site, however, offers you for the first time a robust gateway or digital storefront from which to not only read and download articles, but to comment, share, and interact with the computing community in a meaningful way and in real time without the limitations of page budgets and print schedules.

The new site will be content- and feature-rich with an emphasis on high-quality editorial. Everything found in the print publication will be available via the Web site, but the site will also contain additional news content updated more frequently than is possible in print. A variety of user-generated content, such as the new Expert Blog aptly named the Blog@CACM, will be contributed to by a growing list of distinguished practitioners and researchers. Periodically, the best of those entries and comments will make their way into the print magazine and the result will be a cross-fertilization of content between the print and online *Communications*. So, for those of you who still prefer to see your name appear in print there is another incentive to go online. The new site will also serve as a gateway to some of the most interesting and relevant existing blogs (see Blog Roll) in the computing community and provide links to related content, books, courses, conferences, SIGs, and other resources. The site will also be heavily integrated with the ACM Digital Library, so as to provide a single entry point for searching both *Communications* articles and other articles published by ACM.

It is important to say that the site will not be all things to all people. That is not the intention. But, if you are a regular reader of *Communications* and you are looking for a way to find more high-quality information on advanced computing topics (for practitioners and researchers), we believe this new site will be a great place to start and over time will find its way into your favorites folder and become a highly respected and valuable resource. At least, that is our ultimate goal.

Scott E. Delman, GROUP PUBLISHER

DOI:10.1145/1467247.1467250

Children's Magic Won't Deliver the Semantic Web

To explain the nature of "Ontologies and the Semantic Web" in his contributed article (Dec. 2008), Ian Horrocks, a leading figure behind the theory and practice of Description Logics (DLs), employed analogous characters and language of the fictional Harry Potter children's novels. Notwithstanding the fact this did not help readers not already familiar with Potter or even those, as there may exist a few, who find the novels utterly boring and repetitive, hearing the same story over again in a new guise prompts me to ask: When will such presentations evolve from toy examples into more realistic accounts of larger, complex ontologies? That is, when will the important issue of scalability in the storage, retrieval, and use of large ontologies (millions of concepts, hundreds of millions of roles/attributes, nontrivial reasoning) be addressed?

Horrocks wrote, "A key feature of OWL is its basis in Description Logics, a family of logic-based knowledge-representation formalisms that are descendants of Semantic Networks and KL-ONE but that have a formal semantics based on first-order logic." While this may be true, it could also mislead a neophyte to conclude that DL is somehow the only formalism for representing and using ontologies. This is far from true. There is at least one alternative formalism, also a direct descendant of KL-ONE—Order-Sorted Feature (OSF) constraint logic[a]—that lends itself quite well to the task. Elsewhere, I also covered how various DLs and OSF constraint logics formally relate to one another.[b]

The trouble I see in such publications by influential members of the World Wide Web Consortium (W3C) is that one particular formalism—DL—is being confused with the general issue of formal representation and use of ontologies. It would be like saying Prolog and SLD-Resolution is the only way to do Logic Programming. To some extent, the LP community's insistence on clinging to this "exclusive method" has contributed to the relative disinterest in LP following its development in the 1980s and 1990s. Similarly, DL formalists have built a de facto exclusive reasoning method—Analytic Tableaux—into their formalism so the same causes always result in the same consequences.

Whether the various languages proposed by the W3C are able to fly beyond toy applications has yet to be proved, especially in light of the huge financial investment being poured into the semantic Web. To realize this promise, we must not mistake the tools for the goal. Indeed, while DLs are admittedly one tool among several for representing and using ontologies, the goal is still to make semantic Web ontology languages work, no matter which method is used, as long as it is formal, effective, and efficient on real data. Otherwise, the semantic Web might well end up being built on nothing more than children's magic.

Hassan Aït-Kaci, Vancouver, Canada

a Ait-Kaci, H. Data models as constraint systems: A key to the semantic Web. *Constraint Programming Letters 1* (Nov. 2007), 33–88; www.cs.brown.edu/people/pvh/CPL/Papers/v1/hak.pdf.

b Ait-Kaci, H. Description logic vs. order-sorted feature logic. In *Proceedings of the 20th International Workshop on Description Logics. Lecture Notes in Computer Science.* Springer-Verlag, 2007; sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-250/paper_2.pdf.

Author's Response:

The Harry Potter example was not intended to be representative of realistic application ontologies. As I discussed in the article, such ontologies are often large and complex, making them unsuitable for didactic purposes.

I certainly didn't mean to suggest that DL is the only possible formal basis for an ontology language. However, it is important to agree on the use of some formalism in order to facilitate the exchange and reuse of ontologies and encourage the development of the tools and infrastructure needed for large-scale ontology development and deployment. This is a major success of RDF and OWL; users now have access to a previously undreamt of range and quality of tools and is a major factor in their popularity.

Finally, the W3C standards relate only to the languages themselves, leaving the design and implementation of tools to developers. The OWL standard does not specify any particular reasoning algorithm, and existing OWL/DL reasoners are based variously on (at least) analytic tableau, resolution, hyper-resolution, query rewriting, saturation, and rule-extended triple stores.

Ian Horrocks, Oxford, U.K.

Give Me the Science of Virtualization, Not Buzzwords

The "CTO Roundtable on Virtualization, Parts I and II" moderated by Mache Creeger (Nov. and Dec. 2008) was a rambling discussion filled with vague assertions, buzzwords, and brand names but few clear concepts. The anecdotal discussion touched on cloud computing, late binding, even the terror attacks of 9/11, without clear logical sequence or relationship with deeper (unstated) definitions or principles.

As far as I know, VM is an operating system concept first implemented by IBM 40 years ago on its punched-card-era mainframes (360–67) and commercially available on PCs for at least the past 10 years. VM was invented for essentially the same reasons it is used today: run multiple operating systems on one machine in fully isolated ways. Some of these operating systems may be less reliable than others or may still be under test but are unable to interfere with one another. Even if we are talking about the same thing, the roundtable highlighted none of these basic concepts. VM was widely used within a few years of its earliest implementation. One roundtable participant (in Part I, Nov. 2008) said: "I support virtualization." OK, so I support transistor radios.

To me, this is further confirmation of the fact that IT progress is fast on the surface but slow in terms of basic concepts.

Luigi Logrippo, Gatineau, Québec, Canada

Creeger's Response:

The CTO Roundtables are conversations, not well-defined treatises with clear-cut conclusions. Discussing early-stage adoption of commercial technology involves differences of opinion about definition, best practices, product maturity, and best ways forward. We provide the discussion; the reader decides.

My focus as moderator is commercial benefits and best ways to realize them, conceding that my success varies. Logrippo suggests and I agree we need to do more to extract key ideas and make them more accessible to the reader.

While virtualization goes back more than 40 years, it has gained renewed commercial appeal in the past decade as a better way to provide application services. Overhead, risk, cost, and resulting benefit must be evaluated in the context of the commercial problems being addressed. The goal is not to define virtualization as a new CS technique but address its relatively recent status as an attractive commercial technology. When a panelist supports virtualization, it mean to him its benefits far outweigh its impact on service infrastructure.

Mache Creeger, Head Wrangler, CTO Roundtable Series, Portola Valley, CA

More Legacy from Gates

Michael Cusumano really knows something about Microsoft, and his Viewpoint column "Technology Strategy and Management" on "The Legacy of Bill Gates" (Jan. 2009) is the best popular assessment I've read on the subject. However, for the public to fully understand how Gates affects the world, three more aspects of that legacy must be understood:

Product lock-in. In the marketplace for everyday consumer software, consumers' decisions are overwhelmed by their need for compatibility with popular file formats; all other desirable attributes, including cost, quality, speed, security, ergonomics, simplicity, size, and feature sets, are simply inactivated by this one imperative. Gates understood this network dynamic at the time he founded Microsoft and has pursued it relentlessly ever since. Never before has a popular world market been so tightly constrained by this idea; billions of consumers have thus been deprived of choices through a single mechanism. Paradoxically, this lock on the market happened even as the technical capacity to produce cheap alternative products mushroomed;

Wheels of justice. As a business calculation, Microsoft ignored a court-imposed fine of one million Euros per day every day for three years. This action (as well as others by Microsoft) created a new level of frustration for court systems and represents a phenomenon of corporate behavior that may now need specific new methods of redress. Speed of compliance with court orders is crucial in a marketplace moving as quickly as IT. As long as the wheels of justice turn slower than marketplace evolution, many laws may be reduced to irrelevance; and

Battle against standards. Microsoft is fully aware that open public standards are an impediment to the perpetuation of its monopolies and spends billions to defeat them. Public standards are a pillar of efficiency in free markets, addressing the lock-in problem by solving the compatibility problems, and hence of immense value to consumers. Unfortunately, the tactics in this battle are largely out of the public's view.

Such business behaviors are only casually understood by the public. None are new, but globalization and the extraordinary new arithmetic of marginal costs in the software industry have intensified their effects. Gates elevated each one to the level of boardroom stratagem, using it to prevent the market from becoming as competitive and productive as it could be. It behooves the world to pay as much explicit attention to these things as Gates did and decide if a response is needed. As economies change, our free-market system requires diligent protection from every scheme that suppresses efficient competition.

J. Stephen Judd, Plainsboro, NJ

Deserves More Than an Ad Hominem Response

When columnist Michael Cusumano used the phrase "religious-like responses from the faithful" in his response to a comment (by Ian Joyner, Dec. 2008, concerning his Viewpoint column "Technology Strategy and Management, Sept. 2008) to simply dismiss the comment, it constituted an ad hominem and self-referential attack, not a principled response, and was unworthy of the professional standards ACM is attempting to establish in the new *Communications*.

Rosemary M. Simpson, Providence, RI

Cusumano's Response:

It was quite a rude comment to me, and I reacted to the tone of it. No doubt it is best in such cases to wait awhile before responding. But when a reader criticizes every argument by saying I am simply "anti-Apple," there is not much use in replying point by point. I have had many such encounters with Apple users, given my extensive work on Microsoft and concluded there is indeed such a thing as "the Apple faithful" and a strong element of religiousness to them. But I disagree that I am simply anti-Apple. I have been much more critical of Microsoft and Bill Gates. The main point was that I believe Apple could have become the dominant PC technology had Steve Jobs adopted more of an open "platform" strategy, much as Japan Victor did with VHS, which dominated Beta mainly because of the much greater availability of prerecorded tapes (software) and extensive OEM licensing deals (hardware).

Michael Cusumano, Cambridge, MA

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.

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's increasing involvement in initiatives aimed at ensuring the health of the computing discipline and profession serve to help ACM reach its full potential as a global and diverse society which continues to serve new and unique opportunities for its members.

As part of ACM's overall mission to advance computing as a science and a profession, our invaluable member benefits are designed to help you achieve success by providing you with the resources you need to advance your career and stay at the forefront of the latest technologies.

MEMBER BENEFITS INCLUDE:

- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full access to over 3,000 online courses** from SkillSoft®
- **Full access to 600 online books** from Safari® Books Online, featuring leading publishers, including O'Reilly (Professional Members only)
- **Full access to 500 online books** from Books24x7®
- A subscription to ACM's flagship monthly magazine, ***Communications of the ACM***
- Full member access to the new ***ACM Queue*** website featuring blogs, online discussions and debates, plus video and audio content
- The option to subscribe to the full **ACM Digital Library**
- The **Guide to Computing Literature**, with over one million searchable bibliographic citations
- 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 **Postini** spam filtering
- And much, much more

ACM's worldwide network of over 92,000 members range from students to seasoned professionals and includes many of the leaders in the field. ACM members get access to this network and the advantages that come from their expertise to keep you at the forefront of the technology world.

Please take a moment to consider the value of an ACM membership for your career and your future in the dynamic computing profession.

Sincerely,

Wendy Hall

Wendy Hall

President
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: ACACM28

You can join ACM in several easy ways:

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

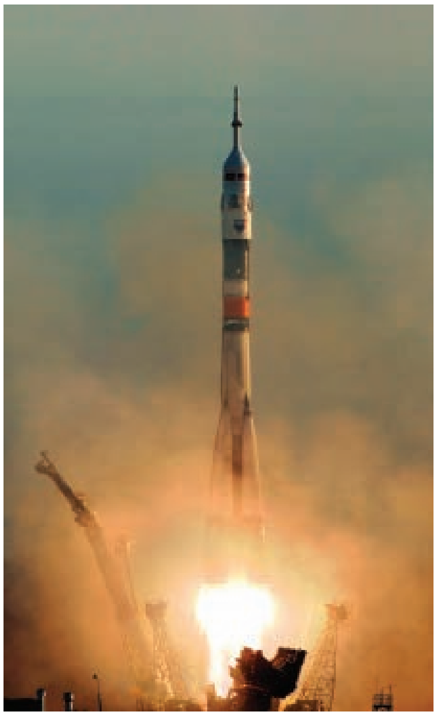
❏ Professional Member Dues ($99 or $198)	$ ________
❏ ACM Digital Library ($99)	$ ________
❏ Student Member Dues ($19, $42, or $62)	$ ________
Total Amount Due	$ ________

Card # Expiration date

Signature

DOI:10.1145/1467247.1467251 David Roman

Prepare for Launch



The task identified by ACM in 2005 has come to fruition. *Communications of the ACM* has been remade both in print and online. The magazine was relaunched in July 2008, and now we are putting the finishing touches on the Web site to launch in March at cacm.acm.org.

To say a Web site is preparing to 'launch' hints at manned spaceflight and adds an element of drama that aggrandizes a site's unveiling. That's unnecessary. The development of the *Communications*' Web site was dramatic enough.

The drama could be found in the faces of ACM managers when they recognized the developers' simpatico braininess. It was in the musings of stakeholders sharing wouldn't-it-be-nice lists, and then realizing that some wishes do come true. It was in the scrupulous attention that *Communications*' Web board members paid to idiosyncratic design details such as fonts, column widths, and breadcrumb trails, and in their elation when they realized their suggestions begat change. And as the launch date drew near, it was in the unbending determination of all parties to work through and past every clash, to square the uncompromising conflict between getting things right and hitting each deadline.

The site is ready, but not finished. That's not to say it is not a complete product. It is. Unlike its predecessor, it delivers a daily dose of news, blogs, and opinion pieces from ACM and from around the Web. It reflects the rich history of *Communications*' 52 years and introduces a new chapter in its editorial scope and global coverage. Indeed, its plentiful content will make you a frequent visitor.

But there are more features, content, and services in the offing. The site's adherence to user-centered design will influence future developments, as will Web trends, user predilections, and hard economics (for more details, see the Publisher's Corner on page 7). The site does and will mirror the membership's diverse and changing interests. Enjoy it!

ACM Member News

WENDY HALL APPOINTED DAME COMMANDER

ACM President Wendy Hall has been appointed Dame Commander of the Order of the British Empire by Queen Elizabeth II for services to science and technology. The appointment was announced by Buckingham Palace as part of the 2009 New Year Honours list.

"Hall is a member of the United Kingdom's Prime Minister's Council for Science and Technology, former president of the British Computer Society, professor of computer science at the University of Southampton, U.K., and a renowned researcher in Web science including multimedia and hypermedia," said ACM CEO John White in a statement. "This honor from The Queen recognizes her service and speaks to Hall's life-long commitment to advancing the field of computer science, as well as supporting her goal of promoting the use of technology to connect people across international boundaries."

Hall is well known throughout the computer science community for her energy and vision, and for being a vocal advocate of women's opportunities in science, engineering, and technology. In addition to her large number of commitments in areas of policy development, she continues to advance new research directions. In 2006, she was one of the founders of the Web Science Research Initiative, along with Sir Tim Berners-Lee, Nigel Shadbolt, and Daniel Weitzner. They are pioneering the new discipline of Web Science, to develop a better understanding of the architectural principles that led to the Web's growth and success, and ensure that these support the Web's future development.

TAPIA 09

The Richard Tapia Celebration of Diversity in Computing conference is being held in Portland, OR, from April 1–4, and will include a technical program, plenary talks, a poster session, a doctoral consortium, a robotics competition, and networking opportunities. For more information, visit tapiaconference.org.

PHOTOGRAPH BY NASA/BILL INGALLS

Science | DOI:10.1145/1467247.1467252

Gregory Goth

Betting on Ideas

Advanced computational models are enabling researchers to create increasingly sophisticated prediction markets.

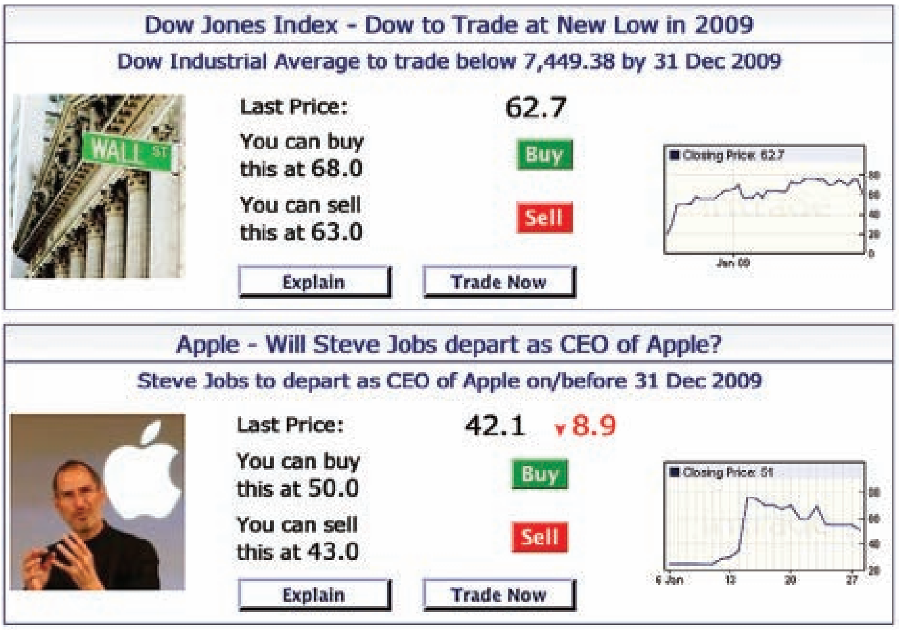
THE U.S. PRESIDENTIAL elections offer social scientists and statisticians many avenues for dissecting the mood of the nation. Among the well-publicized polls and surveys conducted by well-known and well-funded organizations, a lower-key method of capturing the likely outcome of the election—prediction markets—is steadily gaining attention from academic researchers and business leaders for use beyond elections, movie box-office earnings, and sporting contest outcomes.

Like other futures markets, prediction markets offer participants the opportunity to trade on their hunches, the difference being that a prediction market offers payout odds based on aggregate hunches of forthcoming events instead of prices.

Prediction markets are gaining interest because the Internet allows greater worldwide access to them, as well as to the ever-increasing amount of data stored on any topic imaginable (which theoretically allows participants to make more informed predictions, individually and in aggregate). These factors, plus the enormous amount of computing power that will make it possible to instantly calculate exponentially small odds, are stimulating new research on advanced computational models in prediction markets. These models could be capable of analyzing entire events such as the annual NCAA collegiate basketball tournament, which begins a 63-game schedule with 2^{63} possible outcomes by the tournament's end.

"I still think it's a growth area," says David Pennock, a principal researcher at Yahoo!, who is working on expanding the capabilities of prediction market outcomes. "Yes, prediction markets get lots of attention every four years during a presidential election, but every election cycle, they get more attention than they did the previous one. The perception of them is growing, startup companies using prediction markets are emerging, and there are a lot of research questions and industry growth still."

In fact, Pennock says, the U.S. Commodities Futures Trading Commission (CFTC) is considering expanding the use of prediction markets beyond low-budget research functions or "play money" markets to regulated public exchanges



Dow Jones Index - Dow to Trade at New Low in 2009

Dow Industrial Average to trade below 7,449.38 by 31 Dec 2009

Last Price: 62.7

You can buy this at 68.0 — Buy

You can sell this at 63.0 — Sell

Explain | Trade Now

Apple - Will Steve Jobs depart as CEO of Apple?

Steve Jobs to depart as CEO of Apple on/before 31 Dec 2009

Last Price: 42.1 ▼8.9

You can buy this at 50.0 — Buy

You can sell this at 43.0 — Sell

Explain | Trade Now

The world's largest prediction market, Intrade, offers bets on everything from the Academy Awards to whether the Higgs boson particle will be observed before or on a certain date.

SCREENSHOT FROM INTRADE.COM

similar to the world's largest prediction market, Ireland-based Intrade.

"The request for comments was actually very well written and it's clear they understand a lot of the issues," Pennock says. Even if public prediction markets for substantial sums are not approved in the U.S., the markets offer considerable promise for enterprise planners who want the latest information on questions such as a product's likely launch date or revenue projections, and public policy forecasters, who can design markets exempt from CFTC oversight.

Growing opportunities in internal private-sector prediction markets are also revealing divergent philosophies among the markets' designers. Many of the public markets feature price-adjustment algorithms built around answering discrete multiple-choice outcomes, such as which candidate will win an election or if a product will launch in month x, y, or z. However, Mat Fogarty, CEO of prediction markets startup Xpree, says enterprise clients need to address questions expressed as continuous variables, such as a date range in which a product will launch or how many units will sell, and those markets need to feature an intuitive interface that encourages participation among those without a great interest in financial or mathematic complexities. The front end of these new prediction markets, as designed by Xpree, will feature interfaces inspired by computer game design, while the back end will replace multiple-choice algorithms with automated market makers based on Bayesian probability, enabling participants to place bets on a range of options.

Forecasting Events

The pioneering, modern public-policy prediction market, the University of Iowa's Iowa Electronic Markets (IEM), is now 21 years old and still offering new events for traders to forecast. First used in the 1988 U.S. presidential election, the IEM has offered markets on congressional elections, federal monetary policy, and inspired university colleagues to run a prediction market on national influenza infection trends. The IEM's unique design also inspired the latest corporate prediction market, a virtual-money internal market operated by Google.

David Pennock of Yahoo! at a "Prediction Markets: Tapping the Wisdom of Crowds" conference organized by Yahoo!'s Technology Development Group.

IEM steering committee member Thomas Rietz, a professor of finance at the university, says the aggregate zero-risk design of the IEM allows the markets to perfectly reflect the aggregate forecast opinions of its participants. By aggregate zero-risk, Rietz explains that when a trader enters a particular bilateral (either/or) market, he or she must buy one share of each choice, called a bundle, for a total cost of $1. If the trader holds the bundle until the market concludes, there is neither profit nor gain. If the trader guesses the outcome successfully, and sells the losing unit of the bundle to another trader while the market is running, he or she picks up the original $1 bet plus whatever price was agreed upon for the losing share that was sold. If the trader chooses to hold onto the loser and sell the eventual winner, however, they will incur the $1 loss at payout time. At any given time, the number of eventual winning shares and losing shares is equal and held by the traders. So, the university bears no counterparty risk and there is no need to provide hedging margins that irrationally affect outcomes.

The most visible enterprise use of prediction markets is to help companies improve product and process development.

"The price you would be willing to buy or sell for today is your expectation of its value in the future—the prices can be directly interpreted as a forecast," Rietz says. "In ordinary futures markets, there is a long-lasting debate, going back to John Maynard Keynes in the 1930s, over whether prices can legitimately be used as forecasts, and it all hinges on whether or not people demand a return or face a risk in aggregate when they're investing in these contracts."

The enterprise markets are offering intriguing design opportunities, as expressed by Xpree's Fogarty, as well as possible benefits beyond mining collective beliefs of what may make a successful product. The Google prediction market, for example, was examined by Bo Cowgill of Google, Justin Wolfers of the University of Pennsylvania's Wharton School of Business, and Eric Zitzewitz of Dartmouth College as a vehicle for the way information flows within an organization. Prediction markets, they assert, provide employees with incentives for truthful revelation and can capture changes in opinion at a much higher frequency than surveys, allowing one to track how information moves inside an organization and how it responds to external events. Proactive managers can use the analysis of those information flows to reorganize the company, if necessary, says Wolfers.

"A problem for economists is you can't measure information flows, and a market actually kind of makes those flows measurable," Wolfers says. "I would never suggest you set up a prediction market just to learn about the sociology of your organization. But it tracks, and can also change, how organizations operate."

Combinatorial markets allow a wider range of outcomes and a more granular expression of them than traditional bilateral markets.

Better Public Policies

Although Wolfers concedes the most visible enterprise use of prediction markets is to help companies improve product and process development, he also says, "As an economist, I am much more enthusiastic about how prediction markets could help in producing better public policies."

One public policy market that is gaining momentum is the University of Iowa's Iowa Health Prediction Market, funded by a $1.1 million grant from the Robert Wood Johnson Foundation. The market supplies invited healthcare professionals with $100 to begin trading their forecasts on flu activity in the coming season (winners are allowed to spend their trading earnings on professional advancement, thereby reducing public opprobrium about people profiting from others' illness).

Improving the flu markets' utility will entail expanding the regions the markets cover, and also tackling the most challenging computational issues facing prediction market designers—creating combinatorial markets that allow a much wider range of possible outcomes, and more granular expression of them, than the traditional win-lose, bilateral markets such as election markets. Yahoo!'s Pennock is experimenting with multiple examples of these combinatorial markets, which allow both conditional "if" questions and conjunctive "and" questions to be combined in virtually unlimited multiple arrangements.

For the flu market, which Pennock says he has discussed with the Iowa researchers, a combinatorial interface would allow traders to bet on more than the expected severity of outbreaks in one region.

With a combinatorial interface, he says, "you would choose a region of the country and choose a date range, and then also choose an outbreak range. This is a combination of things you think will happen—'In this region, during this time frame, flu outbreak level will be red.' And the market will price it for you."

One enduring research problem on combinatorial markets is mitigating the effects a virtually unlimited spectrum of outcomes will have on creating markets that are so thin in trades they do not serve their purpose of aggregating information.

In such markets, which might bear a resemblance to an enterprise prediction market in that there are not enough participants to provide a statistically valid spread of opinion, Pennock says a market-maker algorithm might serve as a price setter within widely acceptable limits.

"I believe that approximation algorithms will be fine for the market maker, because people don't really care about making bets on things that are incredibly unlikely, like 10^{-6} chance," Pennock says. "But as long as you're betting on something with a 10% chance of happening, we'll be able to approximate pretty quickly with a market-maker price."

Pennock says the continuous increase of computational power is making advanced research into some of these exponentially based markets feasible. "I don't think it would have happened 10 years ago," he says. "The horsepower to do a good approximation is somewhat more recent." C

Gregory Goth is an Oakerville, CT-based writer specializing in science and technology.

Data Mining

Slice, Then Stitch

Researchers at the University of California, Davis, and Lawrence Livermore National Laboratory have developed software that makes the analysis and visualization of large data sets possible without the use of a supercomputer, reports *Technology Review*. The researchers' algorithm slices data into manageable chunks, then stitches it back together, so the data can be manipulated in three dimensions, all on a computer with the power and capacity of an expensive laptop.

The researchers' algorithm offers a method of obtaining structural information about materials, proteins, and fluids, says Attila Gyulassy, the UC Davis researcher who led the project. It allows users to "interactively visualize, rotate, apply different transfer functions, and highlight different aspects of the data," he says.

The software uses a mathematical tool called the Morse-Smale complex, which has been used to extract and visualize elements of large data sets by sorting them into segments that contain mathematically similar features. The Morse-Smale complex has been known for decades, but it normally requires enormous amounts of computer memory.

Gyulassy and his colleagues overcame this memory problem by writing an algorithm that breaks apart a data set before using the Morse-Smale complex, then stitches the blocks back together. As a result, only a small amount of data is needed at each step, so much less data must be stored in memory.

Peter Schröder, a professor of computer science at the California Institute of Technology, notes that memory has been one of the limiting factors for the complex analysis of massive data sets. "You can't even fit the stuff in memory," he says. "But [the researchers] have addressed it."

The researchers plan to release an open source software library this spring to allow researchers to take advantage of the approach, and revise it according to their needs.

Society | DOI:10.1145/1467247.1467254 Leah Hoffmann

Crowd Control

Using crowdsourcing applications, humans around the world are transcribing audio files, conducting market research, and labeling data, for work or pleasure.

THOUGH COMPUTERS HAVE outstripped us in arithmetic and chess, there are still plenty of areas where the human mind excels, such as visual cognition and language processing. And if one mind is good, as the proverb goes, two—or two thousand—are often better. That insight, and its consequences, drew worldwide interest with the 2004 publication of James Surowiecki's best-selling *The Wisdom of Crowds*, which argued that a large group of people are superior at certain types of rational tasks than individuals or even experts.

Now researchers are turning to computers to help us take advantage of our own cognitive abilities and of the wisdom of crowds. Through a distributed problem-solving process variously known as crowdsourcing, human computation, and computer-aided micro-consulting, answers are solicited online to a set of simple, specific questions that computers can't solve. Is this a picture of a fish? Do you like that style of shoe? How many hotels are on St. George's Island, and which ones have Internet access?

The amateur, often anonymous workers who agree to execute these tasks are usually given some sort of social or financial incentive. A few cents might buy the answer to a simple data-labeling task, while a more arduous job like audio transcription could require a couple of dollars. Reposition the task as a game, and many people even "work" for free. Either way, the possibilities—for creating corpuses of annotated data, conducting market research, and more—have both computer scientists and companies excited.

One of the oldest commercial crowdsourcing applications is Amazon's Mechanical Turk. Named after a famous 18th century chess-playing "machine" that was secretly operated by a human, it offers a flexible, Web-based platform for creating and publicizing tasks and distributing micropayments. Since its launch in 2006, Turk has spawned both a vocabulary and a mini-marketplace. Workers, or "Turkers" (there are more than 200,000 in 185,000 countries, according to Amazon), select "Human Intelligence Tasks" (HITs) that match their interests and abilities. Motivations vary. Some work odd hours or at night to generate extra income, while others simply desire a more productive way to kill time online, like solitaire with financial rewards. As in the offline world, more money buys faster results, and Amazon's HIT requesters often experiment to find a pay scale that matches their needs.

Also part of the Turk economy are companies like Dolores Labs and CastingWords, which rely on Amazon's technology to power their own crowdsourcing applications. Dolores Labs, based in San Francisco, posts Turk HITs on behalf of its clients, then filters the answers through custom-built software systems to check for quality and generate meaningful results. Data is ultimately used to perform tasks like filter comment spam, tag data for search engine optimization, and research market trends.

"Many companies don't have the resources to describe tasks, put them up online, and manage the data they get," explains Lukas Biewald, the company's founder and CEO. Nor do they have time for Dolores's extensive quality-control measures, which include creating "test" questions whose answers are already known, checking responses against one another, tracking individual answer histories, and creating a confidence measure with which to weight the resulting data.

Dolores also guides clients through the many variables that are involved in designing a crowdsourced project. How arduous is each task? How quickly are results needed? How would clients like to deal with the statistical outliers that are caught by Dolores' quality-control algorithms? If you're checking user-generated content for pornography, for example, you might err on the side of caution.

According to Biewald's estimates,

Some of the 10,000 sheep created for Aaron Koblin's TheSheepMarket.com by workers for Amazon's Mechanical Turk who were paid .02 cents to "draw a sheep facing to the left."

IMAGE FROM THESHEEPMARKET.COM BY ARON KOBLIN the cost for a crowdsourced project ranges from $2,000 to $4,000 for simple tagging projects to $10,000 to $20,000 for more complex custom applications. Stephen Mechler, managing director of the German crowdsourcing Web site Floxter, which uses its own technologies to handle the mechanics of creating and assigning tasks and compensating workers, calculates that it is 33% less expensive to crowdsource projects like data classification and tagging than to complete them with in-house employees.

Other companies focus their crowdsourcing efforts on specific types of projects. New Mexico-based CastingWords uses Turk to transcribe audio files. Through a propriety algorithm, files are first split into three- to four-minute chunks. Next, Turkers listen to a few seconds of each clip to judge the quality of the recording, which in turn helps determine pay rates for the transcription work. Once each file has been transcribed, a full draft is assembled and sent back to Turk to be graded for consistency and precision, and re-transcribed where necessary. Finally, Turkers edit and polish the transcript to be sent back to the client. Total costs range from $.75 to $2.50 per audio minute, depending on how quickly a client needs the work completed.

Researchers like Carnegie Mellon University computer science professor Luis von Ahn are also finding ways to put crowdsourcing to work. Unlike his corporate peers, von Ahn is unable to pay for the completion of a task, so he relies on social incentives—and tries to make tasks fun. To entice people to manually label a collection of digital images, for instance, von Ahn created the ESP Game, which randomly matches each player with an anonymous partner. Players try to guess which words or phrases their partners (whom they can't communicate with) would use to describe a certain image. Once both players type the same descriptor, a new image appears and the process begins anew. In 2006, Google licensed the idea and created its own version of the game in order to improve image search results.

Since then, von Ahn has developed other games with a purpose to harness the wisdom of crowds. In Peekaboom, for example, one player attempts to guess the word associated with a particular image as the other player slowly reveals it. In fact, designing a game is much like designing an algorithm, as von Ahn has pointed out: "It must be proven correct, its efficiency can be analyzed, a more efficient version can supersede a less efficient one." And since many people are inherently competitive, building a community around each game to recognize outstanding performers helps increase participation, as well.

Since 2007, some 400 million people have helped digitize more than five billion words, says Carnegie Mellon's Luis von Ahn.

ReCAPTCHA, on the other hand, is an attempt to take advantage of a task that millions of people perform in the course of their everyday online lives: solve the ubiquitous character recognition tests known as CAPTCHAs to prove they are human. "I developed reCAPTCHA because I found out that we're wasting 500,000 collective hours each day solving these mindless tasks," says von Ahn. To put that brainpower to use, reCAPTCHA presents users with scanned images from old books and newspapers, which computers have difficulty deciphering. By solving the reCAPTCHA they help digitize the works. Since 2007, some 400 million people have helped digitize more than five billion words, according to von Ahn.

Crowdsourcing's critics claim it is unethical and exploitive, paying pennies or nothing for honest labor (though diligent workers often make close to minimum wage). In a struggling economy, people may grow choosier about the ways they earn extra income. On the other hand, they may also be more interested in blowing off steam on the Internet—and being rewarded with a few extra dollars.

Leah Hoffmann is a Brooklyn-based science and technology writer. Valerie Nygaard, Microsoft, contributed to the development of this article.

Online Social Networks

Adults Get Social

The percentage of Internet users age 55 and older who have a profile on an online social network has quadrupled during the last four years, from 8% in 2005 to 35% in 2008, according to a new survey by the Pew Internet & American Life Project.

Although media coverage has largely focused on how children and young adults use social network sites, adults still comprise the majority of the users of the social network sites because adults make up a larger portion of the U.S. population than teens, 65% of whom use social network sites.

Overall, however, younger adults are much more likely than older adults to use social networks. For instance, 75% of online adults age 18–24 have a profile on a social network; 57% of online adults 25–34 have a profile; 30% of online adults 35–44 have a profile; 19% of online adults 45–54 have a profile; 10% of adults 55–64 have a profile; and only 7% of online adults 65 and older have a profile.

In terms of gender, adult women and men are equally likely to use social networks.

The Pew study also reported that minority groups are more prevalent on social sites than previously expected. It found that 48% of African-American adults and 43% of nonwhite Hispanic adults have a social profile, compared to 31% of white adults.

The personal use of social networks is more prevalent than professional use, both in the orientation of the networks that adults choose to use as well as the reasons they give for using the applications. For instance, 50% of adult users have a profile on MySpace, 22% on Facebook, and 6% on LinkedIn.

The applications are mostly used to explain and maintain personal networks, and most older adults are using them to connect with people they already know, usually to keep up with (89%), make plans with friends (57%), or to make new friends (49%). Other uses include organizing with others for an event, cause or issue; flirting; promoting one's self or work; and making new business contacts.

Technology | DOI:10.1145/1467247.1467253 Kirk L. Kroeker

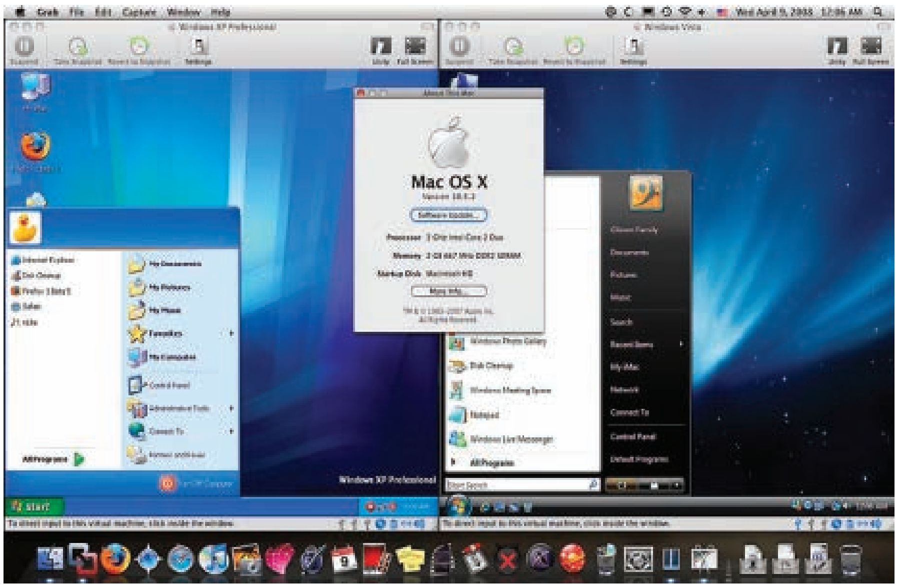
The Evolution of Virtualization

Virtualization is moving out of the data center and making inroads with mobile computing, security, and software delivery.

It's no secret that virtualization, a technology long associated with mainframe computers, has been transforming data centers due to its ability to consolidate hardware resources and reduce energy costs. But in addition to its impact on data centers, virtualization is emerging as a viable technology for smartphones and virtual private networks, as well as being used to reconceive agile and cloud computing.

Over the past decade there has been a great deal of work on improving the performance, enhancing the flexibility, and increasing the manageability of virtualization technologies. Developments in the past five years alone, for example, include the ability to move a running virtual machine, along with its live operating system and applications, to a physical host without major downtime. The industry has also recently witnessed the ability of virtualization to log the actions of a virtual machine in real time, with the purpose of being able to roll back an entire system to an arbitrary point and then roll it forward for debugging or auditing. These and other recent developments have positioned virtualization as a core technology in cloud computing and have facilitated the technology's move to the desktop.

"It's clear that virtualization is here to stay," says Steve Herrod, chief technology officer at VMware. "In the future, we'll look back at the nonvirtualized compute models as we look back at the phonograph and bulky CRTs." But Herrod also says that the industry is far from realizing the full benefits that virtualization can bring to desktops, laptops, and smartphones. "Virtualization is picking up steam rapidly for desktop users, but it has certainly not achieved ubiquity yet," he says. "End users don't want or need to know that virtualization is being used; they want access to their applications, and they want the very rich media experiences that many modern applications offer."


An iMac computer, with VMware Fusion, which enables it to run Windows XP Pro on the left screen, Windows Vista Home on the right, and Mac OS X Leopard in the background.

Screenshot by Christopher Glisson

Arguably, one of the most interesting and novel uses of the technology is on mobile devices, where virtualization enables several new use-cases, such as isolating work and home smartphones on a single physical handset. Gartner predicts that more than 50% of new smartphones will have a virtualization layer by the year 2012. The need for virtualization on smartphones is strong, says Herrod, particularly as these devices become more powerful, as mobile applications become more advanced, and as security becomes a bigger issue. "Just as in the early days of our x86 desktop virtualization efforts, we see many different benefits that will come with this virtualization," says Herrod.

As one example, Herrod cites the substantial testing procedures that every new handset must undergo prior to shipping. Virtualization, he says, will let handset manufacturers test once and deploy on different handsets. For the carriers, Herrod predicts that virtualization will enable a new set of services, such as allowing users to deploy a virtual copy of their mobile data to a newly purchased handset. And for businesses, he says that those who want a single handset for home and work will be able to use different virtual phones. "Their work phone could be restricted to very specific applications and corporate data that is secure and completely isolated from their home phone, where they may have personal information and games," he says. "The more we talk with people about this new area, the more use-cases we find."

Enhanced Security

The notion that one of the strengths of virtualization is its ability to isolate data and applications corresponds to another aspect of the technology that has become increasingly popular. While it might be easy to think of virtualization as adding a software layer that requires additional controls to maintain security, proponents of virtualization argue that it serves the opposite purpose, and instead represents a core enhancement to security. "The only way we know how to get strong isolation is to keep things simple," says Mendel Rosenblum, founder of VMware and a professor of computer science at Stanford University. "And the only way we know how to do that is to have isolation enforced at the lowest level."

Modern operating systems have a high level of functionality—and a corresponding level of complexity and number of potential weaknesses. "I look at virtualization as a step toward getting out of the mess we have in terms of these systems being so insecure," says Rosenblum, who maintains that better security is a natural result of virtualization. Still, he says, it is incumbent on those working on virtualization to build layers that don't make virtualized systems so full of features and complex that they become difficult to secure.

Ian Pratt, founder of XenSource and vice president of advanced products at Citrix, has a similar view of virtualization's relationship to security. "If you look at hypervisors for laptops and phones, it's not about consolidation," he says. "It's about security and being able to secure different partitions on a device."

Citrix is developing software for a model of mobile computing that the company calls "bring your own computer," with the idea being for employees to use their own laptop for securely connecting to the corporate network. In this model, the laptop runs a corporate virtual machine directly on top of a hypervisor rather than in a hosted virtual environment contained by the employee's personal operating system.

"You need to provide very strict isolation between those environments because you really don't trust the personal environment," says Pratt. "It is only through using a hypervisor where you can achieve that strong isolation between those environments."

Like VMware's Herrod, Pratt points to smartphones as one manifestation of this new way of thinking about virtualization and security. In Pratt's example, a handset might have one virtual machine that controls the radio, another that contains all the default software and applications, and a third that operates everything the user downloads and installs. "The whole idea behind this," says Pratt, "is that because you have this strong isolation, no matter what rubbish you download and install on the phone, you are still going to be able to make that 911 call whenever you need it."

With virtualization, people will be able to use both their work phone and home phone on a single handset.

Proponents of virtualization say that, in addition to facilitating new ways of enforcing security, virtualization technologies are leading to new ways of distributing software. "Virtualization not only gives you the ability to manage hardware more effectively," says Rosenblum, "but also allows you to treat the software you're running differently." One way of leveraging virtualization's capabilities is to ship complete packages of running virtual machines rather than having users assemble operating systems and applications themselves, he says. The idea represents a different take on software as a service, a model that obviates the need for users to assemble applications themselves. "It's not like you buy all the separate parts to make a car, but that's what we do with computers," says Rosenblum, who predicts that virtualization will lead to users simply invoking complete, authenticated virtual machines tailored to their particular needs.

Core Challenges

While virtualization is continuing to make inroads in several new areas and

Quantum Computing

Atoms Teleported

A team of scientists from the University of Maryland and the University of Michigan have successfully teleported information between a pair of atoms, housed in separate and enclosed containers, across a distance of one meter, reports *Science*. According to the scientists, this is the first time that information has been teleported between two separate atoms in unconnected containers.

With their protocol, the scientists successfully teleported quantum information between two ytterbium ions, using a method of teleportation in which the ions are stimulated to emit photons and the quantum states are inferred from the color of the emissions. The scientists report that atom-to-atom teleported information can be recovered with perfect accuracy approximately 90% of the time, and they believe that figure can be improved.

"Our system has the potential to form the basis for a large-scale 'quantum repeater' that can network quantum memories over vast distances," says Christopher Monroe, the team leader and a physics professor at the University of Maryland. "Moreover, our methods can be used in conjunction with quantum bit operations to create a key component needed for quantum computation.

"One particularly attractive aspect of our method is that it combines the unique advantages of both photons and atoms," says Monroe. "Photons are ideal for transferring information fast over long distances, whereas atoms offer a valuable medium for long-lived quantum memory. The combination represents an attractive architecture for a 'quantum repeater,' that would allow quantum information to be communicated over much larger distances than can be done with just photons. Also, the teleportation of quantum information in this way could form the basis of a new type of quantum Internet that could outperform any conventional type of classical network for certain tasks."

is leading to speculation about new models of computing, the technology's overhead remains a core challenge. Recent advances in hardware and software have been removing some of the performance concerns associated with virtualization, but the goal is to eliminate the performance gap altogether. "We are not there yet, but what you're going to see is enhancements in processors and other technologies to make the performance gap go away," says Leendert van Doorn, who is a senior fellow at AMD and responsible for AMD's virtualization technology, including the AMD virtualization extensions in the company's latest quad-core Opteron processor, which are designed to reduce the performance overhead of software-based virtualization. "The big problem with virtualization right now is performance guarantees," he says. "If you have a database transaction requirement of a few milliseconds, it is very difficult to provide that guarantee in a virtualized environment."

Still, van Doorn says he is confident that this overhead will be reduced in the coming years with better hardware and software support for virtualization. Currently, overhead in virtualized environments varies from a few percent to upward of 20%, a figure that van Doorn says depends on several factors, including how the hypervisor is implemented and whether the operating system running atop the hypervisor is aware that it is being virtualized. "The Holy Grail is to get near-native performance," he says. "We are getting closer to that goal."

In the future, all new machines might have virtualization capabilities embedded in their firmware.

In addition to the performance issue, there remains the issue of manageability in the data center and elsewhere. "For the next generation, every big software company is working on comprehensive management tools," says van Doorn. The goal is to deal with a massive number of virtual machines and effectively make global optimization decisions for thousands of virtual systems running in data centers or in the hands of a large work force. Sophisticated management tools will be essential in the future imagined by virtualization's proponents, who predict that industry is moving toward a world in which the technology is ubiquitous, and where all new machines will have virtualization capabilities embedded in firmware.

Certainly, says Citrix's Pratt, all servers, desktops, laptops, smartphones, routers, storage arrays, and anything else running software that must be isolated from other applications will be virtualized. The result? "The main noticeable thing will be more trustworthy computing," says Pratt. Echoing this sentiment, Herrod predicts that users won't think about virtualization as a different form of computing. "It will seamlessly fit into our notion of computing," he says, "enabling a much simpler and more productive experience for all of us." C

Based in Los Angeles, **Kirk L. Kroeker** is a freelance editor and writer specializing in science and technology. Steven Hand, Citrix, and Carl Waldspurger, VMware, assisted in the development of this article.

Obituaries

In Memoriam

The world of computer science recently lost two esteemed members: Oliver G. Selfridge, who died at 82, and Ingo Wegener, 57.

Selfridge, whose career included positions at MIT, BBN, and GTE Laboratories, is widely regarded as a leading pioneer in the field of artificial intelligence and the father of machine perception. "In prescient research in the 1950s," says Eric Horvitz, president of the American Association of Artificial Intelligence, "he introduced and tackled key problems that are now well known to machine learning researchers, including the challenges of search and optimization over large parameter spaces, feature definition and selection, dependencies among variables, and unsupervised learning—learning without explicit access to signals about success versus failure."

In 1956, Selfridge, with four colleagues, organized a conference at Dartmouth College that led to the creation of the field of artificial intelligence. And his 1958 paper, "Pandemonium: A Paradigm for Learning," is a classic AI treatise that essentially provides a blueprint for machine learning research.

"The Pandemonium work introduced a distributed model for pattern recognition, where a community of interacting 'demons' or agents with different competencies and functions perform different subtasks that are then combined into final answers or behaviors," Horvitz notes. "Rather than being handcrafted ahead of time and fixed, the agents and their networks of communication could evolve with experience.

"For decades, Oliver communicated an exciting vision where computers would one day learn to infer human intentions and act to assist people without the need for detailed expression of problems," says Horvitz. "Such a vision has evolved to be central in research on human-computer interaction."

Ingo Wegener, a professor of computer science at the Technical University of Dortmund, is well known for his groundbreaking work in complexity theory. He wrote a pair of important monographs, *The Complexity of Boolean Functions* (1987) and *Branching Programs and Binary Decision Diagrams* (2000). In the early 1990s, he worked in the formal analysis of metaheuristics, and his conviction that optimization algorithms based on metaheuristics, like evolutionary algorithms and simulated annealing, should be studied with the methods from the theory of efficient algorithms and complexity theory. Wegener's new, theoretical approach produced a profound understanding of the limitations of such metaheuristics.

Wegener was appointed a member of the German Council of Science and Humanities, the leading scientific advisory committee to the German government, in 2004, and won the Konrad-Zuse-Medal, Germany's most prestigious computer science award, in 2006.

LEFT: PHOTOGRAPH COURTESY OF CAROLINE SELFRIDGE. RIGHT PHOTOGRAPH COURTESY OF INFORMATIONSDIENST WISSENSCHAFT

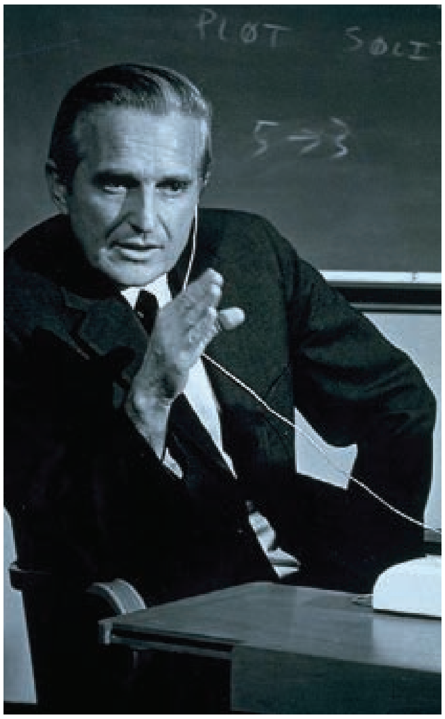
News | DOI:10.1145/1467247.1467255 Karen A. Frenkel

A Difficult, Unforgettable Idea

On the 40th anniversary of Douglas C. Engelbart's "The Mother of All Demos," computer scientists discuss the event's influence—and imagine what could have been.

On December 9, 1968, Douglas C. Engelbart and his Stanford Research Institute (SRI) team demonstrated their latest inventions at the Fall Joint Computer Conference in San Francisco in an event popularly known as "The Mother of All Demos." Engelbart's demonstration included the world debut of the computer mouse, plus the introduction of interactive text, email, teleconferencing and videoconferencing, and hypertext.

But Engelbart, director of SRI's Augmentation Research Center, had lofty aspirations for the system, called NLS (for oNLine System). His goal was to create an integrated system that would "augment human intellect" by facilitating collaboration and bootstrapping—continually improving the improvement process—and thereby help people better the world. NLS, he hoped, would enable a new way of thinking about how humans work, learn, and live together.



Clockwise from top left: A video still of Douglas C. Engelbart during "The Mother of All Demos" in 1968; Engelbart conducting a workshop circa 1967; and a closeup view of the ergonomic keyboard and mouse setup used in the 1968 demonstration.

PHOTOGRAPHS COURTESY OF BOOTSTRAP ALLIANCE

Last December two celebrations—one at the Tech Museum of Innovation in San Jose, and another at Stanford University—commemorated the demonstration's 40th anniversary, and industry luminaries honored Engelbart and his team's achievements, discussed how the event changed their thinking, and examined its impact on computing.

Andries van Dam, a professor of computer science at Brown University, extolled what he had felt back then was so "mind-blowing" about the demo—that it reflected a broad, new way of thinking about design. "It was a huge beautiful suite of tools that allowed a recursive, self-improvement process—very fast progressive refinement cycles that really raised the collective IQ of the group and made the tools more powerful," he said.

However, van Dam was disappointed that the idealism of an integrated system has been lost. "Today we have silos—application programs that mirror the development organizations that produced them" and whose "common denominator is importing and exporting bitmaps," he said. Computing, van Dam suggested, should "go back to the future."

Alan Kay, president of Viewpoints Research Institute, said what most attracted him to Engelbart's goal was to use computers to improve the world. However, people disagreed about what it means to augment intellect. Furthermore, he said, the biggest unsolved problem is how to capture group wisdom and the difficulty of summarizing it.

Kay and van Dam both lamented today's practitioners' lack of curiosity and historical context. "We're incredibly wedged... conceptually, technically, emotionally, and psychologically into a tiny and boring form of computing that is not even utilitarian," said Kay. "I'd be happy to burn the whole thing down and start over again."

Kay said few people objected when browsers were no longer WYSIWYG-capable "because [people] were not sophisticated enough to have the perspective to complain." And van Dam objected to "dumbed-down" links. In the past, "we had fine-grained, bidirectional, tagged links useful for information retrieval and viewing specifications for links and their destinations," he said. "We need to get them back and not just be stuck with URLs."

Kay warned that suboptimal tools can reshape us, and called on attendees to spread Engelbart's vision. "Perhaps the real significance of NLS," he said, "is that it put an idea into the world that is a difficult one, but... it's an idea none of us can forget." C

Based in Manhattan, **Karen A. Frenkel** is a freelance writer and editor specializing in science and technology.

News | DOI:10.1145/1467247.1467256

ACM Fellows Honored

Forty-four men and women are being inducted this year as 2008 ACM Fellows.

The ACM Fellows Program was established in 1993 to recognize and honor outstanding ACM members for their achievements in computer science and information technology and for their significant contributions to the mission of the ACM. The ACM Fellows serve as distinguished colleagues to whom the ACM and its members look for guidance and leadership as the world of information technology evolves.

The men and women honored as Fellows have made critical contributions toward and continue to exhibit extraordinary leadership in the development of the Information Age, and will be inducted at the ACM Awards Banquet on June 27, 2009, in San Diego, CA.

This year's 44 new inductees bring the total number of ACM Fellows to 675 (see www.acm.org/awards/fellows/ for a complete list of ACM Fellows).

ACM Fellows

Martín Abadi, Microsoft Research Silicon Valley/University of California, Santa Cruz
Gregory Abowd, Georgia Institute of Technology
Alexander Aiken, Stanford University
Sanjeev Arora, Princeton University
Hari Balakrishnan, Massachusetts Institute of Technology
William Buxton, Microsoft Research
Kenneth L. Clarkson, IBM Almaden Research Center
Jason (Jingsheng) Cong, University of California at Los Angeles
Perry R. Cook, Princeton University
Stephen A. Cook, University of Toronto
Jack W. Davidson, University of Virginia
Umeshwar Dayal, Hewlett-Packard Laboratories
Xiaotie Deng, City University of Hong Kong
Jose J. Garcia-Luna-Aceves, University of California, Santa Cruz/Palo Alto Research Center
Michel X. Goemans, Massachusetts Institute of Technology
Patrick Hanrahan, Stanford University
Charles H. House, Stanford University MediaX Program
Watts S. Humphrey, SEI, Carnegie Mellon University
Alan C. Kay, Viewpoints Research Institute
Joseph A. Konstan, University of Minnesota
Roy Levin, Microsoft Research Silicon Valley
P. Geoffrey Lowney, Intel Corporation
Jitendra Malik, University of California, Berkeley
Kathryn S. McKinley, The University of Texas at Austin
Bertrand Meyer, ETH Zurich
John C. Mitchell, Stanford University
Joel Moses, Massachusetts Institute of Technology
J. Ian Munro, University of Waterloo
Judith S. Olson, University of California at Irvine
Lawrence C. Paulson, University of Cambridge Computer Laboratory
Hamid Pirahesh, IBM Almaden Research Center
Brian Randell, Newcastle University
Michael K. Reiter, University of North Carolina at Chapel Hill
Jennifer Rexford, Princeton University
Jonathan S. Rose, University of Toronto
Mendel Rosenblum, Stanford University
Rob A. Rutenbar, Carnegie Mellon University
Tuomas Sandholm, Carnegie Mellon University
Vivek Sarkar, Rice University
Mark S. Squillante, IBM Thomas J. Watson Research Center
Per Stenström, Chalmers University of Technology
Madhu Sudan, Massachusetts Institute of Technology
Richard Szeliski, Microsoft Research
Douglas Terry, Microsoft Research Silicon Valley

Call for 2009 ACM Fellows Nominations

The designation "ACM Fellow" may be conferred upon those ACM members who have distinguished themselves by outstanding technical and professional achievements in information technology, who are current professional members of ACM, and have been professional members for the preceding five years. Any professional member of ACM may nominate another member for this distinction.

Nomination information organized by a principal nominator should include excerpts from the candidate's current curriculum vitae, listing selected publications, patents, technical achievements, honors, and other awards; a description of the work of the nominee, drawing attention to the contributions that merit designation as Fellow; and supporting endorsements from five ACM members.

Nominations and endorsements must be submitted to the ACM Fellows Web site by September 1, 2009. For more information about ACM Fellows and other member grades, visit http://awards.acm.org/html/amg_call.cfm.

Call for Submissions and Participation

The 5th International Symposium on Wikis (WikiSym 2009)

WikiSym 2009

The 5th International Symposium on Wikis

Submission deadline: March 27, 2009

Symposium: October 25 – 27, 2009

Symposium location: Orlando, FL, U.S.A.

Symposium Chair: Dirk Riehle, SAP Labs LLC

Program Chair: Amy Bruckman, Georgia Tech

For more information see

http://www.wikisym.org/ws2009

DOI:10.1145/1467247.1467257

Peter J. Denning and Richard D. Riehle

The Profession of IT

Is Software Engineering Engineering?

Software engineering continues to be dogged by claims it is not engineering. Adopting more of a computer-systems view may help.

It is a time of considerable introspection for the computing field. We recognize the need to transcend the time-honored, but narrow image of, "We are programmers." That image conveys no hint of our larger responsibilities as software professionals and limits us in our pursuit of an engineering model for software practice.

The search for an alternative to the programmer image is already a generation old. In 1989 we asked: Are we mathematicians? Scientists? Engineers?[3] We concluded that we are all three. We adopted the term "computing," an analogue to the European "informatics," to avoid bias toward any one label or description.

Today, we want all three faces to be credible in an expanding world. The cases for computing as mathematics and as science appear to be widely accepted outside the field.[1] However, the case for computing as engineering is still disputed by traditional engineers. Computer engineering (the architecture and design of computing machines) is accepted, but software engineering remains controversial.

In this column, we examine reasons for the persistent questions about software engineering and suggest directions to overcome them.

Software engineering may suffer from our habit of paying too little attention to how other engineers do engineering.

Engineering Process

The dictionary defines engineering as the application of scientific and mathematical principles to achieve the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems. When applied to software engineering, this definition calls attention to the importance of science and math principles of computing. Software engineering has also contributed principles for managing complexity in software systems.

Some definitions insist that engineering mobilizes properties of matter and sources of energy in nature. Although software engineering does not directly involve forces of nature, this difference is less important in modern engineering.

The main point of contention is whether the engineering practices for software are able to deliver reliable, dependable, and affordable software. With this in mind, the founders of the software engineering field, at the legendary 1968 NATO conference, proposed that rigorous engineering process in the design and implementation of software would help to overcome the "software crisis."

In its most general form, the "engineering process" consists of a repeated cycle through requirements, specifications, prototypes, and testing. In software engineering, the process models have evolved into several forms that range from highly structured preplanning (waterfalls, spirals, Vs, and CMM) to relatively unstructured agile (XP, SCRUM, Crystal, and evolutionary). No one process is best for every problem.

Despite long experience with these processes, none consistently delivers reliable, dependable, and affordable software systems. Approximately one-third of software projects fail to deliver anything, and another third deliver something workable but not satisfactory. Often, even successful projects took longer than expected and had significant cost overruns. Large systems, which rely on careful preplanning, are routinely obsolescent by the time of delivery years after the design started.[2] Faithful following of a process, by itself, is not enough to achieve the results sought by engineering.

Engineering Practice

Gerald Weinberg once wrote, "If software engineering truly is engineering, then it ought to be able to learn from the evolution of other engineering disciplines." Robert Glass and his colleagues provocatively evaluated how often software engineering literature does this.[4] They concluded that the literature relies heavily on software anecdotes and draws very lightly from other engineering fields. Walter Tichy found that fewer than 50% of the published software engineering papers tested their hypotheses, compared to 90% in most other fields.[8]

So software engineering may suffer from our habit of paying too little attention to how other engineers do engineering. In a recent extensive study of practices engineers expect explicitly or tacitly, Riehle found six we do not do well.[5]

▸ *Predictable outcomes (principle of least surprise).* Engineers believe that unexpected behaviors can be not only costly, but dangerous; consequently, they work hard to build systems whose behavior they can predict. In software engineering, we try to eliminate surprises by deriving rigorous specifications from well-researched requirements, then using tools from program verification and process management to assure that the specifications are met. The ACM Risks Forum documents a seemingly unending series of surprises from systems on which such attention has been lavished. Writing in ACM SIGSOFT in 2005, Riehle suggested a cultural side of this: where researchers and artists have a high tolerance, if not love, for surprises, engineers do everything in their power to eliminate surprises.[6] Many of our software developers have been raised in a research tradition, not an engineering tradition.

▸ *Design metrics, including design to tolerances.* Every branch of modern engineering involves design metrics including allowable stresses, tolerances, performance ranges, structural complexity, and failure probabilities for various conditions. Engineers use these metrics in calculations of risk and in sensitivity analyses. Software engineers do not consistently work with such measures. They tend to use simple retrospective measures such as lines of code or benchmark performance ranges. The challenge is to incorporate more of these traditional engineering design metrics into the software development process. Sangwan gives a successful example.[7]

▸ *Failure tolerance.* Henry Petroski writes, "An idea that unifies all engineering is the concept of failure. Virtually every calculation an engineer performs…is a failure calculation… to provide the limits than cannot be exceeded." There is probably no more important task in engineering than that of risk management. Software engineers could more thoroughly examine and test their engineering solutions for their failure modes, and calculating the risks of all failures identified.

▸ *Separation of design from implementation.* For physical world projects, engineers and architects represent a design with blueprints and hand off implementation to construction specialists. In current practice, software engineers do both, design and build (write the programs). Would separation be a better way?

▸ *Reconciliation of conflicting forces and constraints.* Today's engineers face many trade-offs between conflict-

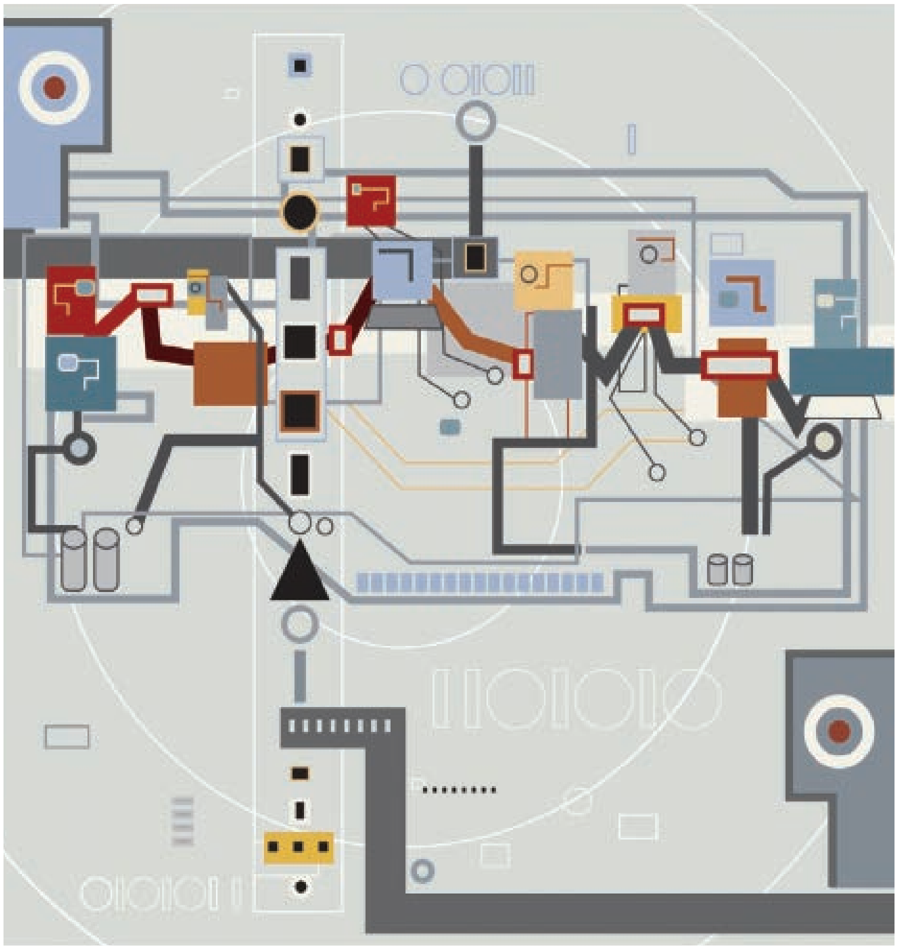


ILLUSTRATION BY CAMILLE CHISHOLM/THEISPOT.COM ing natural forces and a dizzying array of non-technical economic, statutory, societal, and logical constraints. Software engineering is similar except that fewer forces involve the natural world.

▸ *Adapting to changing environments.* Most environments that use computing constantly change and expand. With drawn-out acquisition processes for complex software systems, it is not unusual for the system to be obsolete by the time of delivery. What waste! Mastering evolutionary development is the new challenge.[2]

The System

The problems surrounding the six issues listed here are in large measure the consequence of an overly narrow view of the system for which the software engineer is responsible. Although controlled by software, the system is usually a complex combination of software, hardware, and environment.

Platform independence is an ideal of many software systems. It means that the software should work under a choice of operating systems and computing hardware. To achieve this, all the platform-dependent functions are gathered into a platform interface module; then, porting the system to another platform entails only the building of that module for the new platform. Examples of this are the Basic Input-Output System (BIOS) component of operating systems and the Java Virtual Machine (JVM). When this can be achieved, the software engineer is justified in a software-centric view of the system.

But not all software systems are platform independent. A prominent example is the control system for advanced aircraft. The control system is implemented as a distributed system across many processors throughout the structure where they can be close to sensors and control surfaces. Another example is software in any large system that must constantly adapt in a rapidly changing environment. In these cases the characteristics of the hardware, the interconnections, and the environment continually influence the software design. The software engineer must either know the system well, or must interact well with someone who does. In such cases adding a system engineer to the team will be very important.

We need to encourage system thinking that embraces hardware and user environment as well as software.

Engineering Team

No matter what process engineers use to achieve their system objectives, they must form and manage an engineering team. Much has been written on this topic. Software engineering curricula are getting better at teaching students how to form and work on effective teams, but many have a long way to go.

Every software team has four important roles to fill. These roles can be spread out among several people.

The software architect gathers the requirements and turns them into specifications, seeks an understanding of the entire system and its trade-offs, and develops an architecture plan for the system and its user interfaces.

The software engineer creates a system that best meets the architecture plan. The engineer identifies and addresses conflicts and constraints missed by the architect, and designs controls and feedbacks to address them. The engineer also designs and oversees tests. The engineer must have the experience and knowledge to design an economical and effective solution with a predictable outcome.

The programmer converts the engineering designs into working, tested code. Programmers are problem-solvers in their own right because they must develop efficient, dependable programs for the design. Moreover, anyone who has been a programmer knows how easy it is to make mistakes and how much time and effort are needed to detect and remove mistakes from code. When the software engineer has provided a good specification, with known exceptions predefined and controls clearly delineated, the programmer can work within a model that makes the job of implementation less error-prone.

The project manager is responsible for coordinating all the parts of the team, meeting the schedules, getting the resources, and staying within budgets. The project manager interfaces with the stakeholders, architects, engineers, and programmers to ensure the project produces value for the stakeholders.

In some cases, as noted previously, a systems engineer will also be needed on the team.

Conclusion

We have not arrived at that point in software engineering practice where we can satisfy all the engineering criteria described in this column. We still need more effective tools, better software engineering education, and wider adoption of the most effective practices. Even more, we need to encourage system thinking that embraces hardware and user environment as well as software.

By understanding the fundamental ideas that link all engineering disciplines, we can recognize how those ideas can contribute to better software production. This will help us construct the engineering reference discipline that Glass tells us is missing from our profession. Let us put this controversy to rest. C

References

1. Denning, P. Computing is a natural science. *Commun. ACM 50*, 7 (July 2007), 13–18.
2. Denning, P., Gunderson, C., and Hayes-Roth, R. Evolutionary system development. *Commun. ACM 51*, 12 (Dec. 2008), 29–31.
3. Denning, P. et al. Computing as a discipline. *Commun. ACM 32*, 1 (Jan. 1989), 9–23.
4. Glass, R., Vessey, I., and Ramesh, V. Research in software engineering: An analysis of the literature. *Information and Software Technology 44*, 8 (2002), 491–506.
5. Riehle, R. An Engineering Context for Software Engineering. Ph.D. thesis, 2008; theses.nps.navy.mil/08Sep_Riehle_PhD.pdf.
6. Riehle, R. Engineering on the surprise continuum: As applied to software practice. *ACM SIGSOFT Software Engineering News 30*, 5 (Sept 2005), 1–6.
7. Sangwan, R., Lin, L-P, and Neill, C. Structural complexity in architecture-centric software. *IEEE Computer* (Mar. 2008), 96–99.
8. Tichy, W. Should computer scientists experiment more? *IEEE Computer* (May 1998), 32–40.

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.

Richard Riehle (rdriehle@nps.edu) is a visiting professor at Naval Postgraduate School in Monterey, CA, and is author of numerous articles on software engineering and the popular textbook *Ada Distilled*.

DOI:10.1145/1467247.1467258

Pamela Samuelson

Legally Speaking
When is a "License" Really a Sale?

Can you resell software even if the package says you can't? What are the implications for copyright law of the Quanta decision discussed in the November 2008 column?

ILLUSTRATION BY ALICIA KUBISTA

WHEN YOU PURCHASE a software package, the package will often inform you (or the software will inform you when you install it on your computer), that you are not the "owner" of a copy of it, but only a "licensee" whose entitlement to use the software is subject to certain restrictions. This may include a restriction on transferring your copy of that software to anyone else.

Suppose you ignore the no-transfer restriction and sell the software to someone. Have you breached an enforceable contractual obligation to the software's developer? By transferring the package to someone else, have you infringed copyright or induced the purchaser of the software to infringe copyright? Is the purchaser of the used software an infringer when he loads it on his computer? Is he too bound by the license restrictions?

There is, oddly enough, no definitive court ruling on these questions. In *Vernor v. Autodesk, Inc.*, a judge recently ruled that a purchaser of used software could lawfully sell the package on eBay because he was entitled to the benefits of the "first sale" rule of copyright law. This rule provides that although copyright owners may control distributions of their works to the public, the first sale of a particular copy to the public exhausts their right to control any further distribution of that copy. The rule applies to all transfers of ownership, including gifts or bequests, but not to licenses.

Reinforcing the *Vernor* ruling was *UMG Recordings, Inc. v. Augusto*, in which a judge recently refused to enforce a restrictive legend forbidding recipients of promotional CDs from selling or otherwise transferring the CDs to other people. As in *Vernor*, the court ruled that it was lawful for Augusto to sell the used CDs on eBay under the first sale rule.

UMG has already appealed the *Augusto* ruling to the Ninth Circuit Court of Appeals (which reviews lower court decisions from California and Washington where *Vernor* and *Augusto* were rendered), and Autodesk is likely to appeal as well. I predict that the *Augusto* ruling will be affirmed. *Vernor* is a closer case, but it too may be affirmed unless the Ninth Circuit overturns one of its long-standing precedents.

UMG v. Augusto

Augusto buys and sells promotional CDs that UMG, among others, ship to insiders in the music business in hopes they will listen to the CDs and thereafter promote the music by spreading positive buzz about it. The CD packaging typically states: "This CD is the property of the record company and is licensed to the intended recipient for personal use only. Acceptance of this CD shall constitute an agreement to comply with the terms of the license. Resale or transfer of possession is not allowed and may be punishable under federal and state laws."

Augusto buys promotional CDs from music stores and online auctions and advertises them on eBay. When UMG found out about this practice, it sent Augusto a cease and desist letter, asserting that selling these CDs would infringe its copyrights. UMG made a similar claim to eBay and asked it to suspend Augusto's account. EBay initially did so, but later reinstated the account after Augusto asserted that his sale of these CDs was lawful under the first sale rule.

UMG then sued Augusto for copyright infringement, alleging that the eBay sales infringed its exclusive right to control distribution of its works. The first sale rule did not apply, in UMG's view, because the CDs had been licensed, not sold, to recipients.

Characterizing a transaction as a license does not, however, automatically make it so. The judge in *Augusto* looked to economic realities to see if the transaction was more like a sale or a license.

One important incident of ownership is a right to an unlimited duration of possession, whereas an incident of a license is an expectation that the property will be returned to its owner when the license expires or is breached. Recipients of the CDs seemed to be entitled to keep the CDs, and UMG produced no evidence that it expected to repossess the CDs. UMG could do nothing, moreover, if recipients destroyed these CDs, even though this would extinguish UMG's claimed property rights. Nor were insider recipients of the CDs under any obligation to UMG to promote the music.

One can generally not obtain broader property rights in an artifact than had the person from whom you got it.

The software industry will likely weigh in heavily on the *Augusto* and *Vernor* cases, for the decisions challenge a long-standing industry practice.

The judge concluded that UMG's shipment of the CDs was a gift to the recipients, not a license. The recipients were, therefore, entitled to transfer their ownership interests in the CDs to Augusto under the first sale rule, and Augusto was free to resell the CDs on eBay.

Vernor v. Autodesk

CTA is an architectural firm that bought 10 copies of AutoCAD software. Some years later, it sold four copies of this software to Vernor at an office sale. Vernor has sold some of them already on eBay. Each time Vernor has tried to sell used AutoCAD software on eBay, Autodesk has contacted him and eBay to assert that the sale would infringe its copyrights because the software had been licensed, not sold, to CTA.

Although eBay initially suspended Vernor's account, Vernor told eBay that the resales were lawful under the first sale rule. Autodesk ultimately acquiesced to some earlier resales by Vernor, but after it objected to his most recent effort to sell a copy of AutoCAD, Vernor sought a declaratory judgment that his resale of the software was lawful under the first sale rule. Autodesk moved to dismiss the complaint, arguing that the first sale rule did not apply.

The judge concluded that Vernor could resell the Autodesk software on eBay because the economic realities of the transaction rendered it a "sale." CTA, after all, had made a one-time payment for permanent use of the software, which is typical of sales transactions. Unlike typically licensed property, Autodesk had no interest in return of the software.

The judge relied on the Ninth Circuit's ruling *U.S. v. Wise*. It held that an actress was the owner of a copy of a film, not a licensee, because she had obtained the right to possess it for an indefinite period and without an obligation to return it, even though she had also agreed not to transfer it and to use it only for personal use.

Applying *Wise*, the judge held that Autodesk had sold the software to CTA, and because of this, the first sale rule protected Vernor's resale of the software on eBay.

What Will the Ninth Circuit Do?

Augusto is an easier first sale case because the restrictive legend printed on the CDs resembles one that was printed in a book that the Supreme Court refused to enforce in *Bobbs Merrill Co. v. Straus*, which established the first sale rule in copyright law.

Recipients of the CDs cannot reasonably be understood to have agreed to UMG's restrictive legend. Indeed, they demonstrated their lack of assent to it by selling or giving the CDs away. Because they were free to transfer the CDs to anyone, so was Augusto.

Consumers Union is submitting an amicus curiae (friend of the court) brief in *Augusto* pointing out that if the Ninth Circuit enforces UMG's restrictive legend and rules that Augusto infringes copyright by reselling the CDs on eBay, this precedent would encourage manufacturers of all types of goods embodying some patented or copyrighted innovation to adopt similar restrictive legends. Such a ruling would substantially undermine competition in the marketplace for used goods.

Enforcing UMG's restrictive legend also seems inconsistent with the Supreme Court's recent decision in *Quanta v. LGE Enterprises*. As I explained in my November 2008 column, the Supreme Court ruled that a patent owner's effort to restrict commerce in licensed technologies was inconsistent with patent law's first sale rule. The Court left open the question about whether purchasers of the technologies could be held liable for breaching contractual restrictions, but made clear that they were not patent infringers.

Vernor is a tougher first sale case than *Augusto* for at least two reasons. First, CTA had agreed to abide by terms of the AutoCAD license. More generally, there is a stronger basis for inferring assent to "license" restrictions when a purchaser of a software package clicks "I agree" to terms of a license when installing the software (although it is not clear from the *Vernor* opinion whether CTA had installed the Autodesk programs). Second, the case law on whether the first sale rule applies to mass-marketed software is mixed.

Some judges have been persuaded that software developers should be free to contract as they wish with their customers who may return the software if they find license terms unacceptable. Some defer to the widespread practice in the software industry of licensing software rather than selling it. If customers have agreed to be licensees and the license forbids transfer of the software, moreover, third-party purchasers such as Vernor are arguably incapable of being "owners" of that software. One can generally not obtain broader property rights in an artifact than had the person from whom you got it.

Yet, other judges have agreed with the *Vernor* decision that a one-time payment of money for a package of mass-marketed software that gives the purchaser rights to use software for an unlimited duration should be treated as a sale, even if it may be subject to some restrictions. Unless the Ninth Circuit overrules the *Wise* decision, Vernor may win the right to resell used software on eBay.

It is a separate question whether CTA breached a contractual obligation to Autodesk by transferring the software to Vernor. But even so, should Vernor be bound by the contract's restrictions on transfers? It would seem not since he has not installed the software on his computer and has not agreed to its terms. A fundamental difference between contract rights and intellectual property rights is that the former bind only the parties to the agreement, whereas the latter bind the world. Besides, Autodesk chose to make the license nontransferable, so how could it bind Vernor or his customers? The Ninth Circuit may view Vernor as an ordinary guy trying to make a buck in the used goods market, rather than an infringer of copyrights.

The software industry will likely weigh in heavily on the *Augusto* and *Vernor* cases, for the decisions challenge a long-standing industry practice. (Negotiated licenses will be unaffected if the Ninth Circuit affirms both rulings.) It remains to be seen whether the Ninth Circuit will recognize as legitimate the interests of people like Augusto and Vernor and their customers in the existence of a market for used goods protected by copyright law. C

Pamela Samuelson (pam@law.berkeley.edu) is the Richard M. Sherman Distinguished Professor of Law and Information at the University of California, Berkeley.

DOI:10.1145/1467247.1467259

David A. Patterson

Viewpoint Your Students Are Your Legacy

This Viewpoint boils down into a few magazine pages what I've learned in my 32 years of mentoring Ph.D. students.

ONE OF MY favorite activities is advising, so I was happy to accept the invitation to give advice about giving advice. Some faculty members give new students a list of their expectations and student rights. One student did so well that I asked him if he knew why. He said I gave him helpful guidance upon entering graduate school, when he was eager to hear it. He then told me what I said, which I've been telling to new students ever since:

▸ *Show initiative, for fortune favors the bold.* Don't wait for professors to tell you what to do; if we were good managers, we probably wouldn't be faculty. Explore, challenge assumptions, and don't let lots of prior art discourage you.

▸ *Sink or swim.* We'll offer you what we think are great projects with plenty of potential, and we'll support you the best we can, but it's what you do with the opportunity that makes or breaks your graduate student career.

▸ *Educate your professor.* We're in a fast-moving field, so for us to give you good advice we need to know what you're working on. Teach us!

It Takes a Village to Raise a Child

Advising is simpler if you foster an environment that helps students learn how to become successful researchers. The general goals of the environment should be:

▸ *Acquiring research taste.* Provide ways for students to acquire research taste; in particular, how to identify problems that if solved are more likely to scale and have impact.

▸ *Frequent feedback.* Offer opportunities for students to practice communication skills by presenting to outsiders, to improve their research via honest feedback, to inspire them with earned praise, and to set milestones for their research.

▸ *Foster camaraderie and enthusiasm.* Create a community that provides camaraderie, group learning, mentoring from senior students, and learning from peers to make the whole Ph.D. process more enjoyable.

Meeting these goals is not always easy. I'll describe three techniques that have worked well for me and many Berkeley systems students: team-oriented, multidisciplinary projects; research retreats; and open, collaborative research labs.

Exciting multidisciplinary projects. I try to work with colleagues to create exciting, five-year projects that I would die to work on if I were a graduate student again. We self-assemble into teams of typically two to four faculty members with the right areas of expertise to tackle a challenging and important problem, then recruit 10 to 20 graduate students to work toward building a prototype that demonstrates our proposed solution. The accompanying table shows

Network of Workstations (NOW) group reunion in 2008.

PHOTOGRAPH BY ERIC ANDERSON

the 10 Berkeley projects on which I participated.

The multidisciplinary nature of the project means students gain hands-on knowledge about other areas by working closely with students and faculty in other fields. The experience they gain building the common prototype helps them develop taste in research topics, which in turn helps them pick interesting research topics for their dissertations and later in the rest of their careers.

Group projects create communities where students have others with whom to interact. In particular, the more senior students can mentor the junior ones. Being a Ph.D. student can be a very lonely experience, especially when it comes time to write a dissertation; being part of a larger group can allay those feelings of isolation.

We recently started celebrating the 10-year anniversary of the end of projects. The high participation level at these reunions indicates that these personal ties in such communities remain 10 years later. The accompanying photo shows the Network of Workstations (NOW) group reunion held last year.[a]

Research retreats. Key to the success of these projects, and to the development of Berkeley systems graduate students, has been twice-a-year, three-day retreats where students on the project present their results to one- or two-dozen guests from industry or non-academic labs. These are intensive events, lasting from early breakfast to late-night discussions, although we do take off one afternoon to have some fun. Retreats act as project milestones, with the specter of presenting to outside visitors motivating students to meet the milestones. We close the retreats with an outsider feedback session that offers advice on any aspect of the research. It's surprisingly rare in academia to get frank feedback about research, but who can't benefit from constructive criticism?

Retreats give graduate students two chances per year to give a serious talk and receive advice from experienced researchers outside academia with different experience and perspectives from the faculty on the project. Students are energized when external people care about their work and find it important. When we advisers say something is good, many students will assume we are just acting as cheerleaders or just trying to get them to work harder. I believe interaction with thoughtful colleagues from industry and non-academic labs is vital to acquiring research taste in computer systems by learning to identify critical problems and impactful solutions. Retreats also introduce students to a network of colleagues that may prove useful later in their careers.

Such projects and retreats might be difficult at some places. Building collaborations with local universities and industry can produce many of the same benefits. The key is to get everyone to stay the full time and have people outside your group provide candid feedback. For example, there is an annual Boston Area Architecture workshop involving Brown, Harvard, UMass, Northeast, RPI, and local industry so that their students can cut their teeth in front of a friendly audience and get feedback from outsiders.

We have been doing retreats for 25 years. To my surprise, three years ago we discovered another technique that is becoming just as important to the success of projects and graduate students.

Open collaborative laboratory. We were increasingly seeing people optimize their schedules to avoid disruptions by working from home when they didn't have classes or meetings, since computers and networks were just as fast at home as in the office. The negative global impact of such a local optimization can be thought of as corollary of Metcalf's Law: if the value of a network is proportional to the square of the number of connected users, even a small group leaving a network can significantly decrease its value. This drop in value can in turn cause others to leave, with the negative feedback loop continuing until the network nearly collapses.

In 2006, we experimented by creating a physical office area with contiguous open space for everyone in the project, including the faculty. We hoped that easy access to faculty would draw students to campus and that the open space would inspire innovation by increasing the chances of spontaneous discussions.[1]

The open space makes it very convenient to quickly grab a group of interested people on a moment's notice for a discussion rather than trying to wander around the building or exchange a volley of email messages to schedule a meeting. We have also been surprised to see new students in this space quickly act like senior graduate students. Apparently, easy access to faculty plus

Patterson's research projects.

Years	Title	Professors	Students
1977–1981	X-Tree: A Tree-Structured Multiprocessor	3	12
1980–1984	RISC: Reduced Instruction Set Computer	3	17
1983–1986	SOAR: Smalltalk On A RISC	2	12
1985–1989	SPUR: Symbolic Processing Using RISCs	6	21
1988–1992	RAID: Redundant Array of Inexpensive Disks	3	16
1993–1998	NOW: Network of Workstations	4	25
1997–2002	IRAM: Intelligent RAM	3	12
2001–2005	ROC: Recovery Oriented Computing	2	11
2005–2010	RAD Lab: Reliable Adaptive Distributed Computing Lab	7	30
2007–2012	Par Lab: Parallel Computing Lab	8	40

a Additional photos are included with the version of this Viewpoint available at the *Communications* Web site, cacm.acm.org. The online version has names and group photos for RAID and SPUR reunions and for the most recent Par Lab and RAD Lab retreats.

watching how senior graduate students operate helps new students move up the learning curve quickly.

The research retreats and open space also build esprit de corps, as we play together one afternoon at retreats—for example, skiing, paint ball, and river rafting—and in the lab we collectively watch presidential debates, movies, and big sports events.

The challenge of our open space is then to preserve concentration while enhancing communication,[1] for otherwise people will still stay home. Distractions are reduced with large displays, headphones, and relying on cellphones instead of landline phones; the custom is to make and take calls outside the open space. We also included many small meeting rooms in which to hold vigorous conversations. The result is an open space about as quiet as a library or coffee shop, which is good enough for most to concentrate while encouraging spontaneous communication.

Actual Advising

Clearly, the students who always do well are a joy to meet. I do wonder how much advising you are really doing for them. For those students who need more help, the only thing I can say with confidence after 32 years is that every student is different, and its unlikely there is a single path that works for all. Moreover, there are limits to how much you can change, since students have had at least 20 years of people shaping their personalities before they even meet you. You can tell new students that being a successful researcher is different from being a successful undergraduate student, as they generally have no opinions on the topic when they arrive. For example, it's often a surprise that grades are less important than research, and that they need to learn how to work on their own rather than just follow orders. They also need to find the right balance between learning the literature and starting to build. Clearly, advice changes over time. New students may need a "starter" project, and you give them larger tasks as students mature: reviewing, mentoring, and even helping write proposals.

Here are a half-dozen other topics for advisors, including bolstering confidence, helping with speaking, spending time together, giving quick feedback, counseling them, and acting like a role model.

Bolster confidence. Self-confidence can be a problem for students, especially early in their careers and for some belonging to underrepresented groups, so look for chances for them to succeed. Perhaps it's suggesting a paper they can be lead author on, taking a summer internship at a company that is a good match for their talents, or even having success as a teaching assistant. I have seen even very senior students blossom late in their careers when they have some wins under their belts that everyone recognizes.

Make sure that you praise such students when they do have real success; all of us love praise for a job well done, but some of us need it more than others do. Students learn from criticism as well as praise, just be careful it doesn't deflate potentially fragile egos. I try to remember to phrase critiques as questions—"What do you think about...?"—both orally and in my written comments on papers. I try to include something to praise in all the red ink that I put on a student's paper, but keep in mind that false praise for a mediocre job may hurt more than help.[3]

Practice public speaking. Good work is often lost due to poor presentation, yet giving good talks is a problem for many students. Our culture is that practice talks are good for everyone, so we all do them, including me. We practice answering difficult questions as well as delivering smooth talks to avoid a "deer-in-headlights" incident during the actual talk.

Advising is simpler if you foster an environment that helps students learn how to become successful researchers.

Spend the time. Weekly meetings gives students a chance to talk about what they're working on and forces them to think in advance about how to utilize their time with you. I tell Ph.D. students in their last six months that they have highest priority on my schedule and can meet as often as they want, which helps reduce their anxiety.

Give feedback, quickly and often. I try to review a student paper within a day or two and give my comments for them to read before we meet, which means I am not the bottleneck. Making students write the paper and the guiding them through the revision process teaches them how to write.

Be a trusted counselor. Students may ask for personal advice, perhaps even for serious problems. As they are often far from family and friends, you must be there for them.

You're a role model; act like one. I am struck from parenting two now-grown sons that it's not what you say but what you do that has lasting impact. I bet this lesson applies as well to your academic progeny. Hence, I am conscious that students are always watching what I do, and try to act in ways that I'd like them to emulate later.

For example, my joy of being a professor is obvious to everyone I interact with, whereas I hear that some colleagues at competing universities often complain to their students about how hectic their lives are. Perhaps differing advisor behavior explains why many Berkeley systems students try academia?

Tricks of the Trade

Surely the most traumatic matter for the students is picking the thesis topic, as they believe it determines their careers. Gerald Estrin, who had worked with John von Neumann, was one of my advisors in graduate school. I still remember him telling me: "Every CS Ph.D. student I have seen, including myself, had a least one period when they are convinced that their dissertation topic is utterly worthless." Just retelling this story can help students cope, but look for opportunities to get others to praise their work. Projects and retreats help: there are others to talk to and they get regular feedback on their chosen topic from the outsiders, which can energize those on the lonely trail to a Ph.D. My view now is that it's not the dissertation topic so much as what students do with it.

Here are four pieces of advice for advisors: help if they stumble, aid non-native speakers, try co-advising, and offer lifelong mentoring.

Help if they stumble. Students may underperform not because they lack ability but because they come to think that "good enough" is OK. Have a heart-to-heart discussion where you point this out and ask if they agree, and from now on they're expected to perform to the best of their ability. The book *The One Minute Manager*[2] offers advice on handling such touchy situations successfully for all involved.

One colleague asks students that seem stuck to send him a daily report about their research and progress. Some days it could just summarize a paper or talk, or even "I didn't do anything." He finds that three to four weeks of this often gets them back on track.

When students really stumble in the program and stop making progress, I have had luck with sending them to industry for a six-month leave, as three months may not be enough to do something significant. Twice students have come back fired up knowing what they want to do for their dissertation and, perhaps more importantly, why they want to do it. A third student decided to stay in industry. That was likely a good decision, as I didn't look forward to trying to drag him across the Ph.D. finish line if he didn't return with a greater sense of purpose, and I'm not sure he would have graduated if he wasn't reinvigorated.

Berkeley CS faculty members hold two meetings a year to review the progress and give feedback to all Ph.D. students. Students meet with advisors beforehand to set mutually agreed upon milestones. Hearing others both praise and criticize your students provides a valuable perspective, and collectively we develop ideas on how to help students in need. Reviews also ensure that no student falls through the cracks. Occasionally, after several warnings, we tell students that their progress is so slow that they should drop out. In more than one instance, these letters lit fires under lethargic students and they filed their dissertations soon thereafter.

Group projects create communities where students have others with whom to interact.

Aid non-native speakers. Non-native English speakers can offer another set of challenges. As far as I can tell, they just need practice speaking and writing English. (I wish this need were limited to non-native English speakers!) Strunk and White's *The Elements of Style*[4] is my writing bible, which I share with all my students. Some colleagues have had luck hiring graduate students from other parts of campus to work with CS graduate students to improve their writing. One colleague suggests making sure that if they share an apartment that their roommates don't speak the same language so that they are forced to speak English. I am trying an experiment to improve the diction of an international student by having him take a course outside the university called "Learn to Speak like an American."

Try co-advising. As part of our new open labs, we are also trying joint advising. I hear my co-advisors offer great advice that I wish I'd said, and I hope vice versa. Co-advising also has the benefit that when one advisor is traveling there is someone else to meet with the student. It also makes advising more fun for everyone involved. I believe it works well if the advisors meet with the student simultaneously, so that they give consistent advice. (From my long years of experience in academia, I've learned you get just as much credit whether you are the sole advisor or if you co-advise a student.)

Mentorship doesn't end at graduation. After investing five or six years training an apprentice, it must be worthwhile to spend a little more time after graduation to help him or her succeed. I offer to give a talk at their new institution to give them one last shove in the right direction. Danny Cohen recently asked for advice from Ivan Sutherland—who supervised his 1968 thesis—adding that Danny views advisor is a lifetime job. I agree. I still offer advice to, and receive it from, my former students. (In fact, my former student Mark Hill suggested I write this Viewpoint.)

Advising in Retrospect

When I was finishing my Ph.D., I read a book based on interviews of people talking about their jobs to help decide what I would do next.[5] What I learned from the book was that people were happy with their careers if they designed or built objects that lasted, such as the Empire State Building or the Golden Gate Bridge, or if they shaped people's lives, such as patients or parishioners. Thus, I went into the job of assistant professor with the hypothesis that my long-lasting impact was not the papers but the people.

Thirty-two years later, I can confirm that hypothesis: your main academic legacy is the dozens of students you mentor, not the hundreds of papers you publish. My advice to advisors is to get your students off to a good start, create stimulating research environments, help them acquire research taste, be a good role model, bolster student confidence, teach them to speak well publicly, and help them up if they stumble, for students are the real coins of the academic realm. C

References

1. Allen, T.J. and Henn, G. *The Organization and Architecture of Innovation: Managing the Flow of Technology.* Butterworth-Heinemann, 2006.
2. Blanchard, K.H. and Johnson, S. *The One Minute Manager.* William Morrow, 1982.
3. Carnegie, D. *How to Win Friends and Influence People.* Pocket, 1998.
4. Strunk, W., and White, E.B. *The Elements of Style, 4th ed.* Longman, 1999.
5. Terkel, S. *Working: People Talk About What They Do All Day and How They Feel About What They Do.* Pantheon Books, Random House, New York, 1974.

David A. Patterson (pattrsn@eecs.berkeley.edu) is the Pardee Professor of Computer Science at U.C. Berkeley and is a Fellow and a past president of ACM.

I'd first like to thank former students for advice on this Viewpoint: Remzi Arpaci-Dusseau, Pete Chen, Mike Dahlin, Garth Gibson, and Mark Hill. Additional thanks to Mark Hill for suggesting developing this Viewpoint about Ph.D. advising. The following Berkeley colleagues improved the draft version of this material: Krste Asanovic, Ruzena Bajsky, Armando Fox, Ken Goldberg, Marti Hearst, Joe Hellerstein, Thomas Henzinger, David Hodges, Randy Katz, Jitendra Malik, John Ousterhout, Alberto Sangiovanni-Vincentelli, Ion Stoica, Jonathan Shewchuk, and Alan Smith. Finally, I'd like to thank everyone who worked with me on the projects listed in the table for helping nurture great students.

DOI:10.1145/1467247.1467260

Jeffrey D. Ullman

Viewpoint Advising Students for Success

Some advice for those doing the advising (and what the advisors can learn from the advisees).

No two doctoral students are the same, and the things an advisor needs to do for each vary accordingly. I can look back over my career and see several approaches that work, and one approach that is popular but doesn't really serve the student well. To begin, the goal of the advisor is to teach someone how to become an independent thinker, inventor, and problem-solver. You must take someone barely out of their teenage years and convince them that they can do something that none of the most experienced people in the field have been able to do. And they must do that not only once, but throughout their professional lifetime. Frankly, when I went off to study for my doctorate, I had no idea what writing a thesis entailed; had I known, I never would have gone to graduate school.

What Not to Do

I was a student, and later faculty member, in an electrical engineering department, where the widely held opinion was that the way you wrote a thesis was to read many papers. Look at the last section, where there were always some "open problems." Pick one, and work on it, until you are able to make a little progress. Then write a paper of your own about your progress, and don't forget to include an "open problems" section, where you put in everything you were unable to do.

Unfortunately this approach, still widely practiced today, encourages

PHOTOGRAPH BY HECTOR GARCIA-MOLINA

Students and colleagues attend Jeff Ullman's retirement celebration in 2003.

mediocrity. It gives the illusion that research is about making small increments to someone else's work. But worse, it almost guarantees that after a while, the work is driven by what *can* be solved, rather than what *needs* to be solved. People write papers, and the papers get accepted because they are reviewed by the people who wrote the papers being improved incrementally, but the influence beyond the world of paper-writing is minimal.

The Early Model: Theoretical Theses

In the first years of computer science as an academic discipline, many theses were "theoretical," in the sense that the contribution was mostly pencil-and-paper: theorems, algorithms and the like, rather than software. While much of this work was vulnerable to the problem just described—paper building on paper—it is quite possible for a theoretical thesis to offer a real contribution. For example, even before I joined the Princeton faculty, I had a summer intern at Bell Labs, Ravi Sethi. At that time Ken Thompson and Dennis Ritchie were involved in the Multics project, an operating system for the GE635 computer. This beast was the first to have more than one register in which arithmetic could be done, and the word passed to Ravi and me that they needed techniques to compile code in a way that made best use of several registers. Ravi's thesis was an algorithm for compiling arithmetic expressions using any given number of registers, in the fewest possible steps. This algorithm actually was put into the C compiler for the PDP-11, a few years later.

While Ravi's thesis was "theoretical"—neither of us wrote any code—the work illustrates how I believe any thesis should develop. The work was not based on what some paper left open, but rather on an expressed need: a way to compile expressions using several registers. The big advantage we had was that we were part of an environment that was pushing the frontiers. Had we not been at Bell Labs, it is doubtful we would have realized the problem was worth addressing. We surely could not have read about it in a paper. Even Andrei Ershov, who had previously published the node-numbering scheme we used, only saw it as a way to compile for a one-register machine, and did not suggest in his paper that someone else should look at machines with multiple registers.

When I went off to study for my doctorate, I had no idea what writing a thesis entailed; had I known, I never would have gone to graduate school.

The Ideal Ph.D. Student

The best scenario is that the student tells me what their thesis should be, and carries it out independently. Moreover, their thesis topic is selected because they perceive a need on the part of some "customer." Sergey Brin came closest to this ideal, since he and Larry Page, with no help from me, saw both the need for a better search engine and the key ways that goal could be reached, while students at Stanford. The one missing element: neither of them got their degree; but more about that later.

A close approximation was George Lueker, who came to visit me one day to ask if I had any ideas for a thesis topic. George was not then my student, being enrolled in the Applied Math Program at Princeton. I happened to be reading about chordal graphs that morning, and suggested an algorithm to detect chordality. A year later, he came back and showed me a thesis he had written on *pq*-trees, a data structure that even today has several important applications beyond chordality testing. Several other students have dragged me kicking and screaming to learn a new area, even if I then got involved in selection of their thesis topic. Matt Hecht had me learn about data-flow analysis; Allen Van Gelder did the same with logic programming.

Why does it matter who suggests the thesis topic? We're trying to get young scientists to the point where they can make independent judgments about what is worth working on. There are several decisions to be made: what is worth doing, what is feasible to do, and how do you do it? While an advisor can help with all these things, it is wonderful to meet a student to whom this comes naturally. Another point that I tried not to forget as I grew older was that young people can often see things that those of us who have become set in our ways cannot. Trusting the technical judgment of the young is not a bad strategy.

What Students Need

To make students successful, we need to be ready to provide many services.

Finding customers. As mentioned at the beginning of this Viewpoint, there needs to be an exposure to problems that are at the frontier, and that are needed by a "customer." Sometimes, they can find a customer in industry, as Ravi Sethi did at Bell Labs. Summer internships can be a great opportunity. However, advisors should encourage students to intern at a strong industrial research group, one where the goals are more than minor tweaks to what exists.

Whether the thesis is theoretical or an implemented solution, students need to be guided to understand who will consume their contribution if they are successful. And the answer cannot be "people will read the paper I will write, and they will use the open problems in it to help form their own theses." Especially when dealing with theoretical theses, the chain of consumption may be long, with idea feeding idea, until the payload is delivered. Yet if we let students ignore the question of whether such a chain and payload plausibly exist, we are doing them a disservice.

Walking before you run. Exposure to problems is not enough. Some, although surely not all, Ph.D. students need to convince themselves that they can do something original. Here are a few ideas that have worked:

- One way to give a beginning student practice with the mechanics of research is to think through a small problem yourself, and then propose to a beginning doctoral student that they work on the problem. Since you have a path in mind, it is easy to raise questions that will lead them where they should go, until they have worked through to the solution on their own. A single experience like this is usually enough to get them operating independently.

▸ Try getting the student to make an early transition from reading papers to exploring their own ideas. Certainly, you need to read enough to get the concepts of your field, but after a point, the more you read, the closer your mode of thinking becomes to that of the field at large, and "out of the box" thinking becomes harder. If they produce promising ideas, then of course a more careful literature search must be performed. I've seen enough examples to believe that it is a rare case (although sadly not impossible) where the student's ideas are completely subsumed under what has already been done.

▸ My colleague Hector Garcia-Molina often encourages students to start not by looking for the theoretically optimal solution, but for a simple, easily implementable solution that gets you 90% of the way there. The optimality might be studied later and can form an important part of the thesis.

▸ My colleague John Mitchell reminds us that even after getting past the hurdle of believing one can invent, the thesis can be intimidating because of its large scale. He gets students to focus on writing a single paper (preferably for a conference where they will meet people, not for a journal). After they have written a few papers, building a thesis from them will seem much less intimidating.

Expressing ideas. An advisor must make sure that their students can write clearly. There is little point training students to generate great ideas if they cannot communicate them. It is essential that the advisor reads very carefully and checks every sentence of a student's first attempts at writing. A common situation, and one that must be caught early, is writing that goes into a lot of detail on the easy parts, and gets fuzzy or overly terse when it comes to presenting the hard parts: the proof of a key theorem or the details of a complex algorithm, for example. So an advisor must judge what is hard and be sure that the writing does justice to those parts.[a]

We're trying to get young scientists to the point where they can make independent judgments about what is worth working on.

Fear factor. Yet another common job of the advisor is to teach the student to fail cheerfully and without embarrassment. Not every student has a built-in fear of failure, but many assume it is wrong to attempt something they doubt is possible. Often, the student's model of a "problem" comes from homework, where the solution is certainly known. They are ashamed to report "I didn't get anything done this week," even if it was not for lack of effort. You don't want students to spend a lot of time trying to write a program that takes another program as input and removes all bugs (as a fellow student of mine was once advised by *his* advisor to try), but it is OK to encourage a student to do something ambitious and risky, like finding more bugs than anybody else. In these cases, a vital job of the advisor is getting students to risk their time and effort, and to deal with the case where nothing good results.

Group therapy. A popular technique for encouraging and engaging students is the free lunch. Not only do Ph.D. students benefit, but it can be used to attract undergraduates into the research community. For the past 15 years, I have been privileged to be part of the "Database Group" (now "Infolab") at Stanford, consisting of faculty Gio Wiederhold, Hector Garcia-Molina, Jennifer Widom, our students, staff, and visitors. At regular Friday lunches, students take turns presenting informal talks on their work, and good-natured argument from the floor is the norm. Students get over the fear of defending their ideas in public, as well as benefiting from insights of others. Students also may practice for an upcoming conference talk and receive very detailed suggestions from fellow students. Another important function of the lunch discussion is bonding, facilitated by a social committee to run group events, and by regular trip reports, which serve as a vehicle for learning about one another's lives.

A Newer Model: Project-Oriented Theses

It took many years to reach this point, but it is now fairly routine to have substantial software projects carried out in an academic setting. While there will always be the occasional thesis that is purely "pencil-and-paper," a much more productive approach is to introduce beginning Ph.D. students to a project. Often they enjoy "learning by doing," contributing to the software development, while learning the new notions that are being investigated by the project. Senior students often get the opportunity to help, and even to supervise, junior students.

The best example I've seen of how to use this mode effectively comes from my colleague Jennifer Widom. In a series of innovative projects (semistructured data, stream databases, and now uncertain databases), she has perfected a routine, consisting of:

1. Define a general goal for the research, and get a team of doctoral students working together.
2. Spend a substantial period of time,

a (Aside: While it sounds pedantic at first, you get a huge increase in clarity by chasing the "nonreferential this" from students' writing. Many students (and others) use "this" to refer to a whole concept rather than a noun. For example: "If you turn the sproggle left, it will jam, and the glorp will not be able to move. *This* is why we foo the bar." Now the writer of this prose fully understands about sproggles and glorps, so they know whether we foo the bar because glorps do not move, or because the sproggle jammed. It is important for students to put themselves in the place of their readers, who may be a little shaky on how sproggles and glorps work, and need a more carefully written paragraph. Today, it is not that hard to find a "this" that is nonreferential. Almost all begin sentences, so grepping for '`This`' will find them.)

perhaps 6–12 months, in which the theory and models underlying the problem area are developed. (Jennifer says that this step—making the students part of the planning and modeling—is what distinguishes her approach.)

3. Then, start an implementation project. Get the students working on pieces. The goal of each project is a robust, distributable prototype, not something that can be carried intact to commercialization.

4. Allow students to identify their own aspect of the broader problem area on whose difficulties they will focus. Students develop their own ideas, which form the core of their thesis, and are able to validate the ideas by installing them in the larger system.

It is sad that many research-funding agencies, such as DARPA, have become so "mission-oriented" recently. While it may be possible to support a Ph.D. student doing part of a project implementation, Step 4 is left out; there is no room on the project for a student to explore original work outside the boundaries of the project. For example, I have heard from several independent sources that while the European Union has been supporting "research" generously, the support is sufficiently constrained by concrete deliverables that there is no way to support Step 4 on the projects. In countries where Ph.D. support comes from a state source, this arrangement presents no serious impediment. However, in countries where Ph.D. students are dependent on project support, it becomes hard to train first-rate researchers.

Students and Startups

One of the trickiest decisions an advisor has to make is how to deal with the student who wants to found a startup while they are working on their doctorate. Few people agree with me on this point, but I believe that, unless the startup idea is insane, they should go out and do the startup. My theory is that, while getting a doctorate and entering the research arena is a high calling, it is not the highest possible calling. A startup can have more impact on our lives than a thesis. Moreover, if they miss the opportunity to do a successful startup, then they have lost a great deal. If the startup flops, as many do, they have lost only a few years, and can resume work on a doctorate if they wish.

Sergey Brin never asked me whether or not he should quit the Ph.D. program and found Google, but I would have told him to do so had he asked. Another student, Anand Rajaraman, did ask my advice on this matter when he was about half a year from finishing. I told him to leave and be a founder of Junglee. The venture was quite successful. A few years later he returned to Stanford, started an entirely new thesis topic that abstracted some of what he had learned at Junglee, and is now Dr. Rajaraman.

You don't have to be in Silicon Valley to think about startups. Great ideas can develop anywhere, and a responsible advisor will, when appropriate, present to their students the option that their work might form the basis of a commercial venture. I recall an email message from a student at another school asking the question: "can a piece of work be both a thesis and useful?" When I replied in the affirmative, I was then asked to explain this point to their advisor. That advisor was serving the student poorly, although their attitude seems fairly common. Even in the course of reviewing this Viewpoint, I encountered the view that a piece of technical work is more to be admired if it cannot be commercialized.

Afterword

Of the various things I've done in my career, I am most proud of my 53 Ph.D. students and their academic descendants (see infolab.stanford.edu/~ullman/pub/jdutree.txt; also see the photo appearing on the first page of this Viewpoint). Many have done things I could never do myself, and done so remarkably well. Each has brought unique talents to their work, and it has, for me, been an education just to watch them. I'd like to imagine that I contributed to their success, although I'm pretty sure that the only thing I really did was stay out of their way so they could realize their own potential. C

Jeffrey D. Ullman (ullman@cs.stanford.edu) is the Stanford W. Ascherman Professor of Computer Science (Emeritus) at Stanford University.

For this Viewpoint, I have repurposed some of the ideas of Hector Garcia-Molina, John Mitchell, Jennifer Widom, and Gio Wiederhold, for which I thank them. Additional thanks go to Mark Hill for suggesting developing this Viewpoint about Ph.D. advising.

Calendar of Events

March 15–19
The 2009 ACM Symposium on Applied Computing, Honolulu, HI, Sponsored: SIGAPP, Contact: Sung Y. Shin, Phone: 605-688-6235, Email: sung.shin@sdstate.edu

March 16–18
10th International Symposium on Quality Electronic Design, San Jose, CA, Contact: Tanay Karnik, Phone: 503-712-4179, Email: tanay.karnik@intel.com

March 19–22
Fourth International Conference on Intelligent Computing and Information Systems, Cairo, Egypt, Contact: Mohamed Essam Khalifa, Phone: 20127937560, Email: esskhlalifa@yahoo.com

March 22–25
7th Annual IEEE/ACM International Symposium on Code Generation and Optimization, Seattle, WA, Sponsored: SIGMICRO, SIGPLAN, Contact: David R. Tarditi, Jr., Email: dtarditi@microsoft.com

March 22–27
2009 Spring Simulation Conference, San Diego, CA, Contact: Gabriel A. Wainer, Email: gwainer@cse.carleton.ca

March 23–26
International Conference on Web Information Systems and Technologies, Lisbon, Portugal, Contact: Joaquim B. Filipe, Phone: 351-91-983-3996, Email: jfilipe@insticc.org

March 31–April 1
Second International Workshop on Social Computing, Behavioral Modeling and Prediction, Phoenix, AZ, Contact: Huan Liu, Phone: 480-727-7349, Email: hliu@asu.edu

viewpoints

DOI:10.1145/1467247.1467261

Len Shustek, Editor

Interview

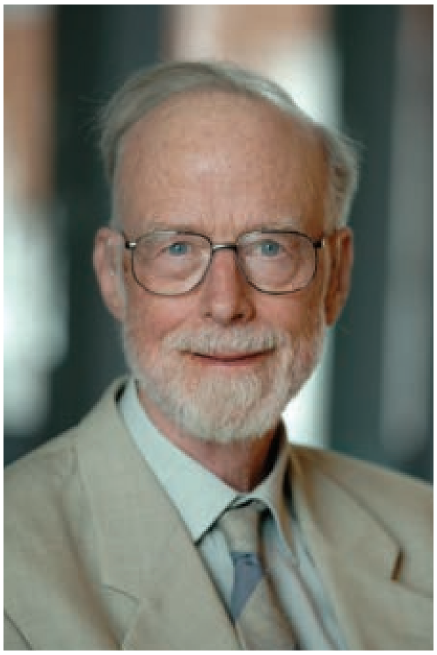
An Interview with C.A.R. Hoare

C.A.R. Hoare, developer of the Quicksort algorithm and a lifelong contributor to the theory and design of programming languages, discusses the practical application of his theoretical ideas.

THE COMPUTER HISTORY Museum has an active program to gather videotaped histories from people who have done pioneering work in this first century of the information age. These tapes are a rich aggregation of stories that are preserved in the collection, transcribed, and made available on the Web to researchers, students, and anyone curious about how invention happens. The oral histories are conversations about people's lives. We want to know about their upbringing, their families, their education, and their jobs. But above all, we want to know how they came to the passion and creativity that leads to innovation.

Presented here are excerpts[a] from an interview with Sir Charles Antony Richard Hoare, a senior researcher at Microsoft Research in Cambridge, U.K. and Emeritus Professor of Computing at Oxford University, conducted in September 2006 by Jonathan P. Bowen, the chairman of Museophile Limited, and Emeritus Professor at London South Bank University.

What did you want to be growing up?
I thought I would like to be a writer. I didn't know quite what I was going to be writing, but at school I was a rather studious and uncommunicative child, and so everybody called me "Professor." I found the works of Bernard Shaw very inspiring. He's of course an iconoclast, so he would appeal to an adolescent. Also Bertrand Russell, who wrote on social matters as well as philosophical and mathematical matters.



PHOTOGRAPH COURTESY OF MICROSOFT RESEARCH

What was your first exposure to computers?
I began thinking about computers as a sort of philosophical possibility during my undergraduate course at Oxford University. I took an interest in mathematical logic, which is the basis of the formal treatment of computer programming. I was sufficiently interested that one of my few job interviews was with the British Steel just after I finished my university course in 1956. I was attracted by their use of computers to control a steel milling line. A little later I attended an interview at Leo Computers Ltd. in London, who were building their own computers to look after the clerical operations of their restaurant chain. But I didn't follow up on either of those prospects of employment.

What was the first program you wrote?
In 1958 I attended a course in Mercury Autocode, which was the programming language used on a computer that Oxford University was just purchasing from Ferranti. I wrote a program that solved a two-person game using a technique which I found in a book on game theory by von Neumann and Morgenstern. I don't know whether it worked or not. It certainly ran to the end, but I forgot to put in any check on whether the answers it produced were correct, and the calculations were too difficult for me to do by hand afterward.

What was programming like in those days?
Very different from today. The programs were all prepared on punched cards or paper tape. It might take a day to get them punched up from the cod-

a Oral histories are not scripted, and a transcript of casual speech is very different from what one would write. I have taken the liberty of editing and reordering freely for presentation. For the original transcript, see http://archive.computerhistory.org/search/oh/.
—Len Shustek

ing sheets, and then they were submitted to a computer maybe the following day. It would take a long time, if there were any faults in the program, to find out where they were.

How did you come to live in Russia?
I did national service, which was compulsory in those days, in the Royal Navy studying modern military Russian. I used to know the names of all the parts of a ship in Russian, even if I didn't know what the actual parts of the ship were. Later I continued my graduate career as a visiting student at Moscow State University for a year.

The 1960s were very exciting times in Russia, especially after the U.S. spy plane was shot down. I felt quite free, and no political problems obtruded. But our Russian friends were very suspicious of each other. We learned quite early on that you never introduce one Russian friend to another, because each of them thinks the other one is the informer. We knew that our rooms were bugged, so we would never talk about Russian friends inside our own rooms.

You developed the famous Quicksort algorithm at about this time. Why?
The National Physical Laboratory was starting a project for the automatic translation of Russian into English, and they offered me a job. I met several of the people in Moscow who were working on machine translation, and I wrote my first published article, in Russian, in a journal called *Machine Translation*.

In those days the dictionary in which you had to look up in order to translate from Russian to English was stored on a long magnetic tape in alphabetical order. Therefore it paid to sort the words of the sentence into the same alphabetical order before consulting the dictionary, so that you could look up all the words in the sentence on a single pass of the magnetic tape.

I thought with my knowledge of Mercury Autocode, I'll be able to think up how I would conduct this preliminary sort. After a few moments I thought of the obvious algorithm, which is now called bubble sort, and rejected that because it was obviously rather slow. I thought of Quicksort as the second thing. It didn't occur to me that this was anything very difficult. It was all an interesting exercise in programming. I think Quicksort is the only really interesting algorithm that I ever developed.

I think Quicksort is the only really interesting algorithm that I've ever developed.

Where did you work after returning to England?
I met my future employers in Russia. I was an interpreter at an exhibition in Moscow, where Elliott Brothers, which at that time made small scientific computers, were exhibiting and selling their computer in Moscow. They offered me employment when I came back, with an additional 100 pounds a year on my salary because I knew Russian. I never had a formal interview.

What did you work on at Elliott?
They were embarking on the design of a new and very much faster computer, and they thought they would celebrate by inventing a new language to program it in. As a recent employee, with six months experience, I was put to designing the language. Fortunately I happened to see a copy of the *Report on the Algorithmic Language Algol 60*, and I was able to recommend to the company to implement that, rather than inventing a language of their own. That proved a very good commercial decision. And a good personal one, because I eventually married Jill, the other programmer who came to work on the same project. She had experience writing a compiler before, which in those days was quite unusual, and she was a much better-disciplined programmer than I ever was.

Was Algol well defined?
The syntax was formally defined. The grammar of the language was written up in a way that was, I think, completely unambiguous. The semantics was a little less formally defined. It used ordinary English to describe what the effect of executing a program would be. But it was very well written by Peter Naur, and it was sufficient to enable us to write a compiler without ever consulting the original designers of the language. And it was sufficient for programmers in the language to write programs, which in the end actually ran on our compiler, without ever consulting us or the original designers of the language. It was a really very remarkable achievement—rather beyond what maybe we can achieve these days in the design of languages.

Did you collaborate with other compiler writers?
We didn't correspond with other people writing compilers, even for Algol, in those days. We didn't know each other. There was no real scientific community that one could join to talk over problems with other people who encountered the same problems. We worked pretty well on our own.

After moving to Queen's University in Belfast in 1968, you wrote a very important paper on the axiomatic approach to computer programming, now known as "Hoare Logic."
I was interested, as indeed many people were at that time, in making good the perceived deficiency of the Algol report: that while the syntax was extremely carefully and formally defined, the semantics was left a little vaguer. We were pursuing the goal of trying to get an equally good formalization of the semantics of the language, a goal that I think still is pretty advanced and maybe beyond our grasp.

I put forward the view that we didn't want the specification to be too precise. We didn't want the specification of a programming language to concentrate in too much detail on the way in which the programs were executed, but rather we should set limits on the uncertainty of the execution of the programs, to allow different implementations to implement the language in different ways. In those days the word lengths and the arithmetic of all of the computers was different. Based on the ideas of mathematical logic that I studied at university, I put forward a set of axioms that describe the properties of the implementation without describing exactly how it worked. It would be possible, I hoped, to state those properties sufficiently precisely that programmers would be able to write programs using only those properties, and leave the implementers the freedom to implement the language in different ways, but at the same time taking responsibility for the fact that their implementation actually satisfied the properties that the program was relying on.

I haven't abandoned this idea, but it didn't turn out to be very popular among language designers.

You then turned to "structured programming," and collaborated with Edsger Dijkstra and Ole-Johan Dahl on an important book.
I met Dijkstra and Dahl at a working conference in 1972 on formal language definition. Dijkstra was the other person writing an Algol compiler at the same time as I was, and Dahl was inventing a new simulation language called Simula, in which he introduced the ideas of object-oriented programming that would later have a great influence on programming and programming languages. All three of us had written draft monographs on our favorite topics: one by Dijkstra called "Notes on Structured Programming," one by Dahl on hierarchical program structures, and my own notes on data structuring. I thought it would be interesting to collect these three together and publish them as a single book.

I spent quite some time trying to understand what Dahl had written. He was a very brilliant but very dense writer. I had great fun trying to simplify some of his really brilliant ideas on how to structure programs in the large. I did not work on Dijkstra's material; that was pretty clear and well written.

What was your involvement with Pascal?
Pascal was the language designed as a teaching language by my friend Niklaus Wirth, after the language we had designed together in the early 1960s had not been recommended. We used to meet quite frequently to talk about the design. My research on applying the axiomatic method to programming language semantics had made quite considerable progress by then, and I thought it was time to see whether I could tackle a complete language using this style of definition. I managed to do the easy bits, but there were still quite a lot of challenges left over, which we just omitted from the definition of the language.

When did you start to look at monitors for operating systems?
In 1972 I organized a conference in Belfast where we assembled quite a brilliant group of scientists to talk about operating system techniques. I was interested in exploring the ideas of from an axiomatic point of view. Could I define axioms that would enable people to safely write concurrent programs in the same way as they can write sequential programs today? We devoted an afternoon to discussing the emerging ideas of monitors; I and Per Brinch Hansen were the main people to contribute to that discussion.

How did you come to move from monitors to communicating sequential processes?
The idea of a communicating process is that instead of calling its components as subroutine, or a method, you'd actually communicate the values that you wanted to transmit to it by some input or output channel, and it would communicate its results back by a similar medium. The reason for this was a technological advance in the hardware: the advent of the microprocessor, a cheap and small machine with not very much store, but capable of communicating with other microprocessors of a similar nature along wires. It was easy to predict that the best way of making a large and fast computer would be to assemble a large number of very cheap microprocessors together, and allow them to cooperate with each other on a single task by communicating along wires that connected them. For this, a new architecture of programs would be appropriate, and perhaps a new language for expressing the programs. That was how the communicating sequential processes came to take a leading role in my subsequent research.

It was easy to predict that the best way of making a large and fast computer would be to assemble a large number of very cheap microprocessors together.

You've always been interested in the connection between theory and practice.
My move back to Oxford was partially motivated by my idea to study the Oxford ideas on the semantics of programming languages again. I hoped that it would be possible, using the Oxford techniques of defining semantics, to clarify the exact meaning of communicating processes in a way that would be helpful to people writing programs in that idiom.

It was a great loss that Christopher Strachey had recently died. I took over his chair at Oxford, quite literally sitting in his chair and sitting at his desk. I happened to open the drawer, and come across a final report on one of his research projects, in which he put forward his ideals for keeping the theory and practice of programming and computer science very much in step with each other. He said the theory could easily become sterile, and the practice could easily become ad hoc and unreliable, if you weren't able to keep one firmly based on the other, and the other firmly studying problems with the practical uses of computers.

Have there been successes using formal methods in practice?
At a keynote lecture for the British Computer Society I talked a bit about formalization and verification, and put forward a conjecture, fairly tentatively, that maybe the time was right to scale these things up by trial use

in industry. A senior director from IBM at Hursley came up to me after the lecture and invited me to put my commitment where my mouth was, and do something in collaboration with IBM. That made me gulp a bit, because I had the impression that IBM had produced some pretty complicated software, and this really would be a challenge. There was a good chance that we would fall flat on our faces. But you can't turn down an opportunity like that. So some colleagues and I set to work, and actually produced some very useful analyses for them using the Z notation of Jean-Raymond Abrial to help them with an ongoing project for restructuring and rewriting parts of their popular customer information software system, CICS. That work eventually led to a Queen's award for technology.

The Transputer was another example of a very practical application of your theoretical ideas.
The INMOS Transputer was as embodiment of the ideas that I described earlier, of building microprocessors that could communicate with each other along wires that would stretch between their terminals. The founder had the vision that the CSP ideas were ripe for industrial exploitation, and he made that the basis of the language for programming Transputers, which was called Occam.

When they came to develop the second version of their Transputer that had a floating- point unit attached, my colleagues Bill Roscoe and Jeff Barrett at Oxford actually used formal models of communicating processes, and techniques of program verification, to check that their designs for the implementation of the IEEE floating-point specification were in fact correct. The company estimated it enabled them to deliver the hardware one year earlier than would otherwise have happened. They applied for and won a Queen's award for technological achievement, in conjunction with Oxford University Computing Laboratory, for that achievement. It still stands out as one of the first applications of formal methods to hardware design.

I expect the future to be as wonderful as the past has been. There's still an enormous amount of interesting work to do.

What projects are you working on at Microsoft?
One of them is the pursuit of my lifetime goal of verification of programs. I was very interested in the ideas of assertions, which had been put forward by Bob Floyd and before; already in 1947 the idea of an assertion was described by Alan Turing in a lecture he gave to the London Mathematical Society. This idea of assertions lies at the very basis of proving programs correct, and at the very basis of my ideas for defining the semantics of programming languages. I already knew when I entered academic life way back in 1968 that these ideas would be unlikely to find commercial exploitation, really, throughout my academic career. I could look forward to 30 years of research uninterrupted by anybody who actually wanted to apply its results.

I thought that when I retired, it would be very interesting to see whether the positive side of my prediction would also come true, that these ideas would begin to be applied. And indeed, even since I've joined Microsoft in 1999, I've seen quite a bit of expansion in their use of assertions and other techniques for improving confidence in the reliability of programs. There are program analysis tools now that stop far short of actually proving correctness of programs, but they're very good at detecting certain kinds of errors. Some quite dangerous errors, which make the software vulnerable to intrusions and virus attacks, have been detected and removed as a result of the use of formal techniques of program analysis.

The idea of verifying computer programs is still an idea for the future, although there are beginnings of using scientific technology to make the programs more reliable. The full functional verification of a computer program against formally specified requirements is still something we have to look forward to in the future. But the progress that we've made has really been quite spectacular in the last 10 years.

My other project is concurrency. I've set myself the challenge of understanding and formalizing methods for programming concurrent programs where the programs actually share the store of the same computer, rather than being executed on distinct computers as they are in the communicating sequential process architecture. Again, the motivation for studying this different form of concurrency is the advance of hardware technology: it now appears that the only way in which processors can get faster is for them to include more processing units on the same chip, and share the same store.

I'd always felt that parallel programs that actually shared main memory, and could interleave their actions at a very fine level of granularity—just a single memory access—were far too difficult for me. I could see no real prospect of working out a theory that would help people to write correct programs to exploit this capability. I thought it would be interesting to try again, and see whether the experience in formalization that has been built up over the last 20 or 30 years could be applied effectively to this extremely complicated form of programming.

What is on the horizon for computer science?
I expect the future to be as wonderful as the past has been. There's still an enormous amount of interesting work to do. As far as the fundamental science is concerned, we still certainly do not know how to prove programs correct. We need a lot of steady progress in this area, which one can foresee, and a lot of breakthroughs where people suddenly find there's a simple way to do something that everybody hitherto has thought to be far too difficult. C

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

DOI:10.1145/1467247.1467262

Smarter, more powerful scripting languages will improve game performance while making gameplay development more efficient.

BY WALKER WHITE, CHRISTOPH KOCH, JOHANNES GEHRKE, AND ALAN DEMERS

Better Scripts, Better Games

The video game industry earned $8.85 billion in revenue in 2007, almost as much as movies made at the box office. Much of this revenue was generated by blockbuster titles created by large groups of people. Though large development teams are not unheard of in the software industry, game studios tend to have unique collections of developers. Software engineers make up a relatively small portion of the game development team, while the majority of the team consists of content creators such as artists, musicians, and designers.

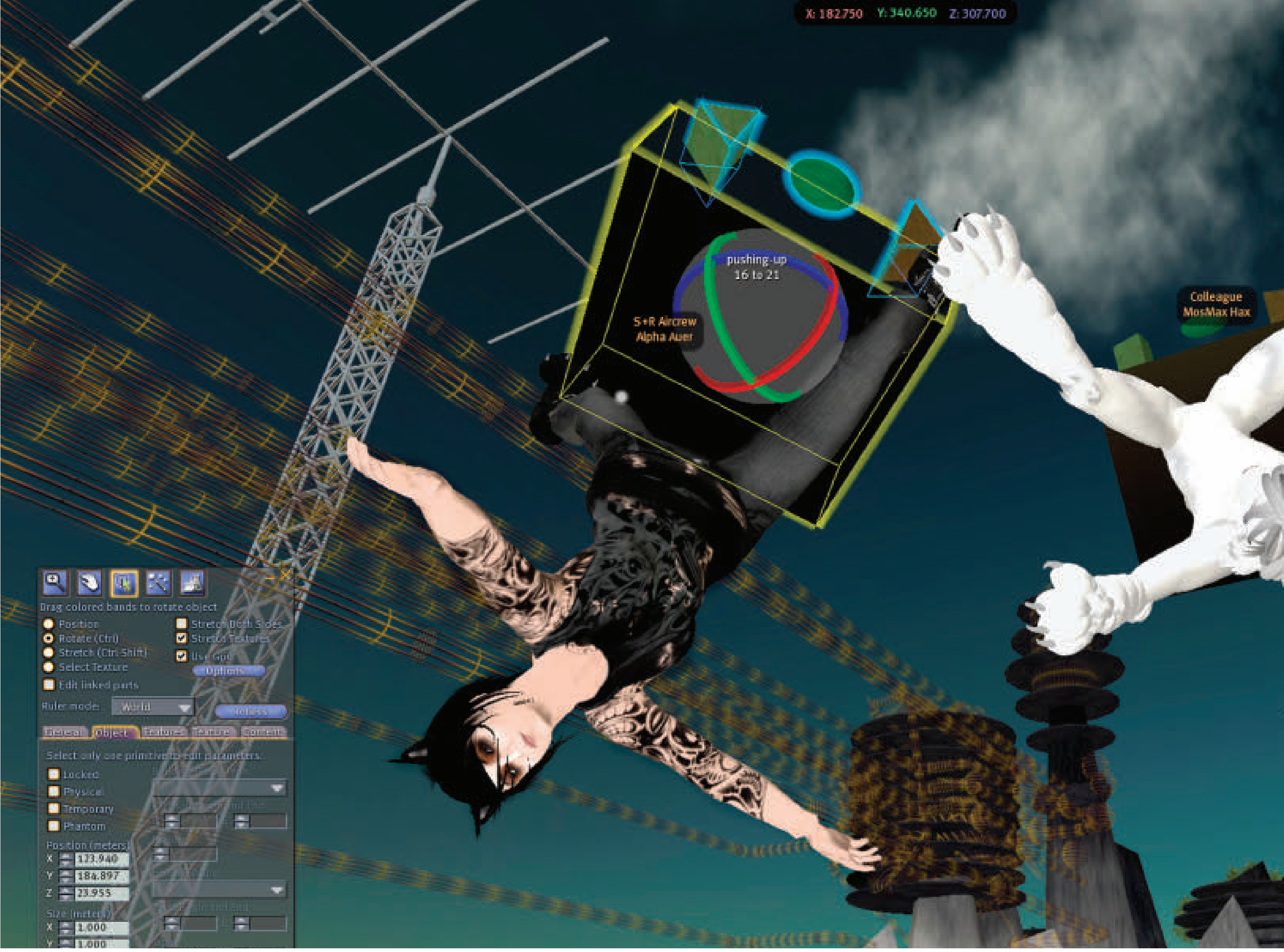
Since content creation is such a major part of game development, game studios spend many resources developing tools to integrate content into their software. For example, entry-level programmers typically make tools to allow artists to manage assets or to allow designers to place challenges and rewards in the game. These tools export information in a format usable by the software engineers, either as auto-generated code or as standardized data files.

This content-creation pipeline is not very well understood, and each studio has its own philosophy and set of tools. Many tools are taken from, or developed in coordination with, the film industry. Unlike film, however, games need to be interactive. Player actions require visual feedback; game characters should react to player choices. Adding interactive features typically requires some form of programming. These features are also a form of artistic content, and game studios would prefer they be created by designers—developers who understand how the player will interact with the game, and what makes it fun—rather than software engineers.

The idea of game software as artistic content has led many game studios to split their software developers into two groups. *Software engineers* work on technical aspects of the game that will be reused over multiple titles. They work on core technology such as animation, networking, or motion planning, and they build the tools that make up the content-creation pipeline. *Gameplay programmers*, on the other hand, create the behavior specific to a single game. Part designer, part programmer, they implement and tune the interactive features that challenge and reward the player.

The gameplay programmer should produce fun, not complex, algorithms. Game studios design their programming workflow to relieve gameplay programmers of any technical burdens that keep them from producing fun. Often this involves an iterative process between the gameplay programmers and the engineers. The gameplay programmers develop feature prototypes to play-test before adding them to the game. The software engineers then use these feature prototypes to design support libraries, which are used to build another round of prototypes. This is an effective workflow, but game companies are always looking for ways to speed up or even automate this process.

In addition to supporting the interaction between gameplay programmers and software engineers, the studios are always looking for ways to integrate the designers into the programming process. Designers often



In this Second Life photograph avatars Alpha Auer and MosMax Hax explore a pose stand that allows users to program poses and run two scripts. One script cycles through the poses, and the other one makes the pose stand invisible/visible.

SECOND LIFE PHOTOGRAPH COURTESY OF ALPHA AUER AND MOSMAX HAX

have very little programming experience, but they have the best intuitions for how the game should play. Thus, studios want tools that allow designers, if not actually to program behavior, at least to fine-tune the parameters behind it.

The Role of Scripting Languages

Many game studios rely on scripting languages to enable gameplay programmers and designers to program parts of their games. These languages allow developers to easily specify how an object or character is supposed to behave, without having to worry about how to integrate this behavior into the game itself. Scripting languages are particularly important for massively multiplayer games where any piece of code must interact with multiple subsystems, from the application layer to the networking layer to the database.

User-created content is another reason for games to support scripting. Open-ended virtual worlds such as Second Life have made player scripting a common topic of conversation. Even before that, games had a long tradition of player-developed *mods*. Given tools—either official or third party—to modify the data files that came with the game, players have been able to create completely new experiences. Generally, modding has been seen as a way to extend the lifespan of older games. In some cases, however, it can create completely new games: the commercially successful Counter-Strike was a player modification of the game Half-Life and relied heavily on scripting features present in its parent game.

Scripting languages allow players to modify game behavior without access to the code base. Just as important, they provide a sandbox that—unlike a traditional programming language—limits the types of behavior the player can introduce. If the game has a multiplayer component, the game developers do not want players creating scripts to give themselves an undue advantage. Overly powerful scripting languages have facilitated many of the bots—automated players performing repetitive tasks—that currently populate massively multiplayer games. Sandboxing can even be useful in-house. By limiting the types of behaviors that their designers can create, the studios can reduce the number of bugs that they can introduce—bugs that cost valuable time to find and eliminate.

The Need for Game-Specific Scripting Languages

The foremost criterion for a scripting language is that it should make gameplay development fast and efficient. Often game objects—rocks, plants, or even intelligent characters—share many common attributes. Game scripting languages are often part of IDEs (such as the one shown in Figure 1) that provide forms for quickly modifying these attributes. The scripting languages themselves, however, are fairly conventional. Many companies use traditional scripting languages such as Lua or Python for scripting. Even companies that design their own languages usually stick with traditional format and control structures. Little effort has been spent tailoring these scripting languages for games.

One of the major problems with traditional scripting languages is that the programmer must be explicitly aware of low-level processing issues that have little to do with gameplay. Performance is a classic example of such a low-level issue. Animation frame rate is so important to developers that they optimize by counting the number of multiplies or adds in their code. This type of analysis is beyond the skill of most designers, however. Furthermore, existing languages provide almost no tools to help designers improve script performance.

Designers must also take performance into account when creating content. If the game runs too slowly, they may be forced to reduce the number of objects in the game, which in turn can significantly alter the playing experience. This is what occurred when The Sims was ported to consoles. In this game, a player indirectly controls a character (Sim) by purchasing furniture or other possessions for it. Each piece of furniture is scripted to advertise its capabilities to the Sim periodically. The Sim then compares these capabilities with its needs in order to determine its next action. Furniture does not exist in isolation, however; a couch in front of a television is much more versatile than one alone in a room. Therefore, pieces of furniture also periodically poll the other furniture in the room to update their capabilities. As each piece of furniture may communicate with other pieces of furniture, the cost of processing a room can grow quadratically with the number of objects in the room. When the title was ported to consoles, the performance issue became so pronounced that the designers had to introduce a "feng shui meter" to prevent players from filling rooms with too many possessions.

Game developers have many techniques available to them for improving performance. Spatial indexes are one popular way of handling interactions between game objects at less than quadratic cost. Parallel execution is another possibility; many games are embarrassingly parallel, and developers leverage this fact for multicore CPUs and distributed multiplayer environments. These techniques are beyond the skill of the typical game designer, however, and are left to the software engineers.

Another low-level issue with scripting languages is the lack of transaction support for massively multiplayer games. Individual scripts are often executed concurrently, particularly in massively multiplayer games, so designers need some form of transaction to avoid inconsistent updates to the game state. Indeed, script-level concurrency violations are one of the major causes of bugs in multiplayer environments.

To make scripting easier for designers, we have to provide them with simple tools for addressing these low-level issues. None of these problems is really new; many programming languages have been developed over the years to address them, but most of these languages make programming more difficult, not easier. Fortunately, designers do not need an arbitrary scripting language; they just need a language that helps them write games.

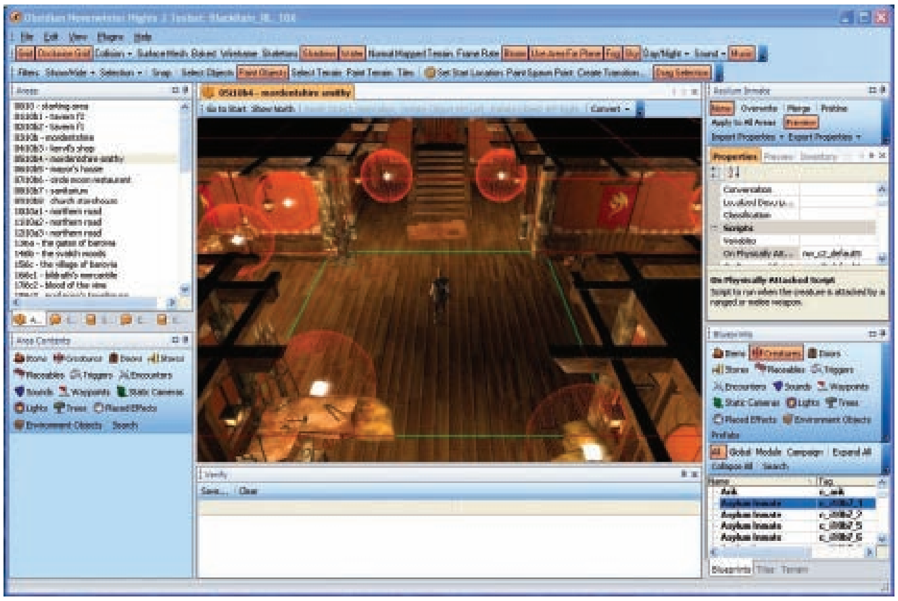


Figure 1. The Neverwinter Nights 2 toolset is an extensive IDE that allows users to create new content for the game.

From Patterns to Language Features

Despite these problems, games are being developed. Game developers have come up with many ideas that, if not complete solutions, do ameliorate the problems. These ideas typically come in the form of programming patterns that have proven over time to be successful. Though developers use these programming patterns in creating game behavior, the scripting languages usually do not support them explicitly. One of the reasons object-oriented programming languages have been so successful is that object-oriented programming patterns existed long before the languages that supported them. Similarly, by examining existing programming practices in game development, we can design scripting languages that require very little retraining of developers. The challenge in developing a scripting language is identifying those patterns and creating language features to support them most effectively.

The State-Effect Pattern

One popular pattern in game development is the *state-effect pattern*. Every game consists of a long-running simulation loop. The responsiveness of the game to player input depends entirely on the speed at which the simulation loop can be processed. In the state-effect pattern, each iteration of the simulation loop consists of two phases: *effect* and *update*. In the effect phase, each game object selects an action and determines individually the effects of this action. In the update phase, all the effects are combined and update the current state of the game to create the new state for the next iteration of the simulation loop.

Because of these two phases, we can separate the attributes of game objects into *states* and *effects*. State attributes represent the snapshot of the world after the last iteration of the simulation loop. They are altered only in the update phase and are read-only in the effect phase. Effect attributes, on the other hand, contain the new actions of the game objects, and the state of the game is updated with effects during the update phase. Because interactions between game objects are logically simultaneous, effect values are never read until the update phase. Hence, effect values are, in some sense, write-only during the effect phase.

Game physics provides many examples of this pattern. At the beginning of the simulation loop, each game object has a current position and velocity recorded as state attributes. To compute the new velocity, each object computes the vector sum of all of the forces acting upon it, such as collisions, gravity, or friction. In other words, the force attribute may be written to multiple times during the simulation loop, but it is never read until all of the force values have been summed together at the end of the loop. The example in Figure 2 illustrates the use of the state-effect pattern to simulate objects moving about in a potential field. The variable `force` is an effect in this calculation. During the effect phase we only increment its value and never read it to determine control flow. Whereas most implementations would read the old value of `force` to perform this increment, this is not necessary; we could also gather all of these force values in a list and add them together at the end of the effect phase.

Most of the time, game developers use the state-effect pattern to manually design high-performance algorithms for very specific cases. That is because it has several properties that allow them to significantly enhance the performance of the simulation loop. The effect phase can be parallelized since the effect assignments do not influence each other. The update phase can also be parallelized since it consists only of the aggregation of effects and updates to state variables. This does not need to be done by hand; if the scripting language knew which attributes were state attributes and which were effect attributes, it could perform much of this parallelization automatically, even in scripts written by inexperienced designers. This is similar to what Google achieves with its Sawzall language and the MapReduce pattern; special aggregate variables perform much the same function as effect attributes, and the language allows programmers at Google to process data without any knowledge of how the program is being parallelized.[1]

Automatic parallelization is an example of an alternative execution model; the game runs the script using a control flow that is different from the one specified by the programmer. Since the simulation loop logically processes all of the game objects simultaneously, we can process them in any order, provided that we always produce the same outcome. Thus, alternative execution models are among the easiest ways of optimizing game scripts. Another unusual execution model is used by the SGL scripting language, which is being developed at Cornell University.[2] This language is based on the observation that game scripts written in the state-effect pattern can often be optimized and processed with database techniques. The script compiler gathers all of the scripts together and converts them into a single in-memory query plan. Instead of using explicit threads, it constructs a data pipeline that allows the code to be parallelized in natural ways. Many of these data pipelines are similar to the ones that game programmers create when they program on the graphics processing unit, except that these are generated automatically.

Figure 2: Example of the state-effect pattern.

```
// Outer simulation loop
for each timestep {

  // Compute effects for all for each particle o {
     o.effectPhase();
  }


// Update state for all for each particle o {
  o.updatePhase() ;
  }

}

// State variables
vector position, velocity;
scalar q, damping, mass;

// Effect variables
vector force;

// Read state, write effects
  effectPhase() {
     for each particle p {
        r = position-this.p.position;
        s = ((this.q*p.q)/(r.magnitude())^3;
        force += s*r;
     }
  }

// Read and write state, read effects
updatePhase() {
  velocity = damping*velocity+force/mass;
}
```

The Restricted Iteration Pattern

Iteration is another common source of problems in game development. Allowing arbitrary iteration can quickly lead

Figure 3: Example of the restricted iteration pattern.

```
//===================================================================
// Nudge items and units within a given rect, so that they can fi nd
// locations where they can peacefully coexist
function NudgeObjectsInRect takes rect nudgeArea returns nothing
  local group g

  set g = CreateGroup()
  call GroupEnumUnitsInRect(g, nudgeArea, null)
  call ForGroup(g, function NudgeUnitsInRectEnum)
  call DestroyGroup(g)

  call EnumItemsInRect(nudgeArea, null, function NudgeItemsInRectEnum)
endfunction
```

to significant performance degradation of the simulation loop. Iteration can be even more dangerous in the hands of inexperienced designers. During the development of City of Heroes, Cryptic Studios discovered that many of the scripts had interdependencies that produced hard-to-find infinite loops. To prevent this, the developers removed unbounded iteration from the scripting language.

Although this was a fairly drastic solution, most games do not need arbitrary iteration in their scripts. The scripts just need to perform a computation over a *finite set of objects*; such scripts follow the *restricted iteration pattern*, which obviously guarantees termination on all loops. In addition, it may enable code analysis and compile-time code transformations that improve performance. For example, SGL can take nested loops that produce quadratic behavior and generate an index structure from them;[2] it then replaces the nested loops with a single loop that performs lookups into that index.

Examples of the restricted iteration pattern appear throughout the scripts in Warcraft III, a real-time strategy game that has to process armies of individual units. The `NudgeObjectsInRect` script in Figure 3 appears in the Blizzard.j file. This function takes a rectangle and loops through all of the military units that appear in that rectangle; in that loop, it uses the function `NudgeUnitsInRectEnum` to push units apart so that there is a minimum distance between pairs of units.

All the operations in this script are external functions provided by the software engineers. The scripting language is not aware that these functions implement the equivalent of a `for-each` loop (a loop over a fixed set of objects); otherwise, the compiler would be able to perform loop optimizations on it. Given the number of times this pattern appears in the Warcraft III scripts, this could result in significant performance improvements.

Concurrency Patterns

Iteration is not the only case in which developers could benefit from alternative control structures. Many games execute scripts in parallel, which requires scriptwriters to be cognizant of concurrency issues. As an example, consider inventory management in online games, a notoriously problematic scenario, with consistency violations resulting in lost or duplicated objects. Consider the following simple script written to put an item in a container such as a sack or a backpack:

```
// Test a container, and
  insert an object if okay
success = TestPutItem(me,
container, item)
if (!success):
  Bail()
else:
  PutItemInContainer(item,
  container)
```

This script tests if a container has the capacity to hold an item, then adds the item if there is space. Nothing in the script says that this action must be executed atomically, so in a distributed or concurrent setting, the container could fill up between the time it is tested and the time the item is added to the container. Obviously, this could be eliminated by the addition of locks or synchronization primitives to the scripting language. Locks can be expensive and error-prone, however, so game developers like to avoid them if at all possible. They are particularly dangerous in the hands of designers.

Additionally, lock-based synchronization is incompatible with the state-effect pattern. In the state-effect pattern, the state of the container consists of the contents at the end of the last iteration of the simulation loop, while an effect attribute is used to gather the items being added to the container. Effect variables cannot be read, even with locks, so the script cannot test for conflicting items being added simultaneously.

Instead of trying to solve this problem with traditional concurrency approaches, it is best to step back and understand what the programmer is trying to do in this pattern. The programmer wants to update an object, but under some conditions this update may result in an inconsistent state. The function `TestPutItem` defines which states are consistent. If the language knew this was the consistency function for `PutItemInContainer`, it could delay the check to ensure consistency without a lock. The language could first gather all of the items to be added to the container and then use the consistency check to place as many as the container can hold. In some cases, the language could even place multiple objects with a single consistency check.

Of course, this approach does not solve arbitrary problems with parallel execution, but game companies use languages with almost no concurrency support, and they rely on coding conventions to limit consistency errors. Adding features that provide concurrency guarantees for the more common design patterns in games would allow the game developers to trust their scriptwriters with a wider variety of scripts, increasing their artistic freedom.

Game-Aware Runtimes

Language features provide the runtime with clues on how best to execute the code, but some games have properties outside of the scripting language that the runtime can also leverage. For example, the right optimization strategy for a set of scripts depends on the current state of the game. If the game is controlling a large army marching toward an enemy, then the game should optimize movement of soldiers; on the other hand, if the army is guarding against an attack, the game should optimize individual perception. Games often have a small number of these high-level states, and changes between them happen relatively slowly. If the runtime can recognize which state the game is in, it can switch to an optimized execution plan and improve performance.

To some degree, game developers already take advantage of this fact in their performance tuning. Currently they log runs of the game during playtesting, and later data-mine these logs for recurring patterns. If these patterns are easy to detect, developers can take advantage of them. This type of optimization, however, is very difficult for designers or for players developing user-created content. Ideally, a game-aware runtime would have some knowledge of common patterns and be able to adjust for them automatically.

Performance is not the only reason for the runtime to monitor how the game changes over time; it is also useful for debugging. Debugging a game is not as simple as stepping through a single script. Each object is scripted individually, and these scripts can interact with one another in subtle ways. An incorrect data value in one script may be the result of an error in a completely different script. In addition, many errors are the result of user input that is not always easy to reproduce. A script designer needs some way of visualizing which scripts modify which objects and how these objects change over time. This is an application of data provenance, which is an active area of development in the field of scientific computation. Like designers, the scientists targeted by data provenance tools often have little programming experience; instead, the provenance techniques model the way they naturally think about the data. As yet, no game scripting language supports data provenance.

Data provenance is even more important if the script runtime has an unusual execution model. In the previous script to place items in a container, efficient execution involved reordering portions of the script. Instead of having the programmer debug the scripts in an execution model that is different from the one in which the bug appeared, it is best to give him or her a higher-level visualization of how that bug might have occurred.

Game-aware runtimes are more difficult to implement than language features. Language features can often be implemented piecemeal; as programming patterns are identified, new language features can be added without adversely affecting the old. Runtimes, once architected, can be very interdependent and difficult to change. For example, any changes to the order in which operations are processed will affect the debugger. Thus, while languages can have an attitude of "see what works," runtimes need to be well understood from the beginning.

By examining existing programming practices in game development, we can design scripting languages that require very little retraining of developers. The challenge in developing a scripting language is identifying those patterns and creating language features to support them most effectively.

Conclusion

Scripting languages are an integral part of both game development and modding, and their design has huge impact on both correctness and performance of the resulting game. Game developers earn money from the titles that they publish, not the engineering problems that they solve. Therefore, anything that reduces technical challenges for the developers and allows them to create more content is a welcome innovation. Advances in design patterns and scripting languages will influence the way games are programmed for years to come.

References

1. Dean, J. and Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan. 2008): 107–113; doi.acm.org/10.1145/1327452.1327492.
2. White, W., Sowell, B., Gehrke, J., and Demers, A. Declarative processing for computer games. In *Proceedings of the 2008 ACM SIGGRAPH Sandbox Symposium*; doi.acm.org/10.1145/1401843.1401847.

Walker White is the director of the Game Design Initiative, an interdisciplinary undergraduate program training students in the design and development of computer games, at Cornell University, Ithaca, NY.

Christoph Koch is an associate professor of computer science at Cornell University, Ithaca, NY.

Johannes Gehrke is an associate professor in the department of computer science at Cornell University, Ithaca, NY. He co-authored *Database Management Systems* (McGraw-Hill, 2002), currently in its third edition.

Al Demers is a principal research scientist in the department of computer science at Cornell University. His current work focuses on scalability and data management for computer games and virtual worlds.

A previous version of this article appeared in the November/December 2008 issue of *ACM Queue*.

DOI:10.1145/1467247.1467263

Designed for concurrency from the ground up, the Erlang language can be a valuable tool to help solve concurrent problems.

BY JIM LARSON

Erlang for Concurrent Programming

Erlang is a language developed to let mere mortals write, test, deploy, and debug fault-tolerant concurrent software.[a] Developed at the Swedish telecom company Ericsson in the late 1980s, it started as a platform for developing soft real-time software for managing phone switches.[1] It has since been open sourced and ported to several common platforms, finding a natural fit not only in distributed Internet server applications, but also in graphical user interfaces and ordinary batch applications.

Erlang's minimal set of concurrency primitives, together with its rich and well-used libraries, give guidance to anyone trying to design a concurrent program. Erlang provides an effective platform for concurrent programming for the following reasons:

- The language, the standard libraries (Open Telecom Platform, or OTP), and the tools have been designed from ground up for supporting concurrency.
- There are only a few concurrency primitives, so it's easy to reason about the behavior of programs (though there are limits to how easy this can ever be).
- The implementation makes the simple primitives fast and scalable, and makes effective use of modern multicore hardware, eliminating the need for more complex mechanisms.
- The execution model eliminates some classes of errors from unsynchronized access to shared state—or at least makes these errors more noticeable.
- The model of concurrency is natural to think about and requires no mathematical sophistication.
- The environment makes failures detectable and recoverable, making it possible to deploy a less-than-perfect system in the field that can nonetheless maintain high availability.
- The concurrency model maps naturally to distributed deployments.

This article introduces the Erlang language and shows how it can be used in practice to implement concurrent programs correctly and quickly.

Sequential Erlang

Erlang is built from a small number of sequential programming types and concepts, and an even smaller number of concurrent programming types and concepts. Those who want a full introduction can find several excellent tutorials on the Web,[b] but the following examples (required by functional programming union regulations) should convey the essentials.

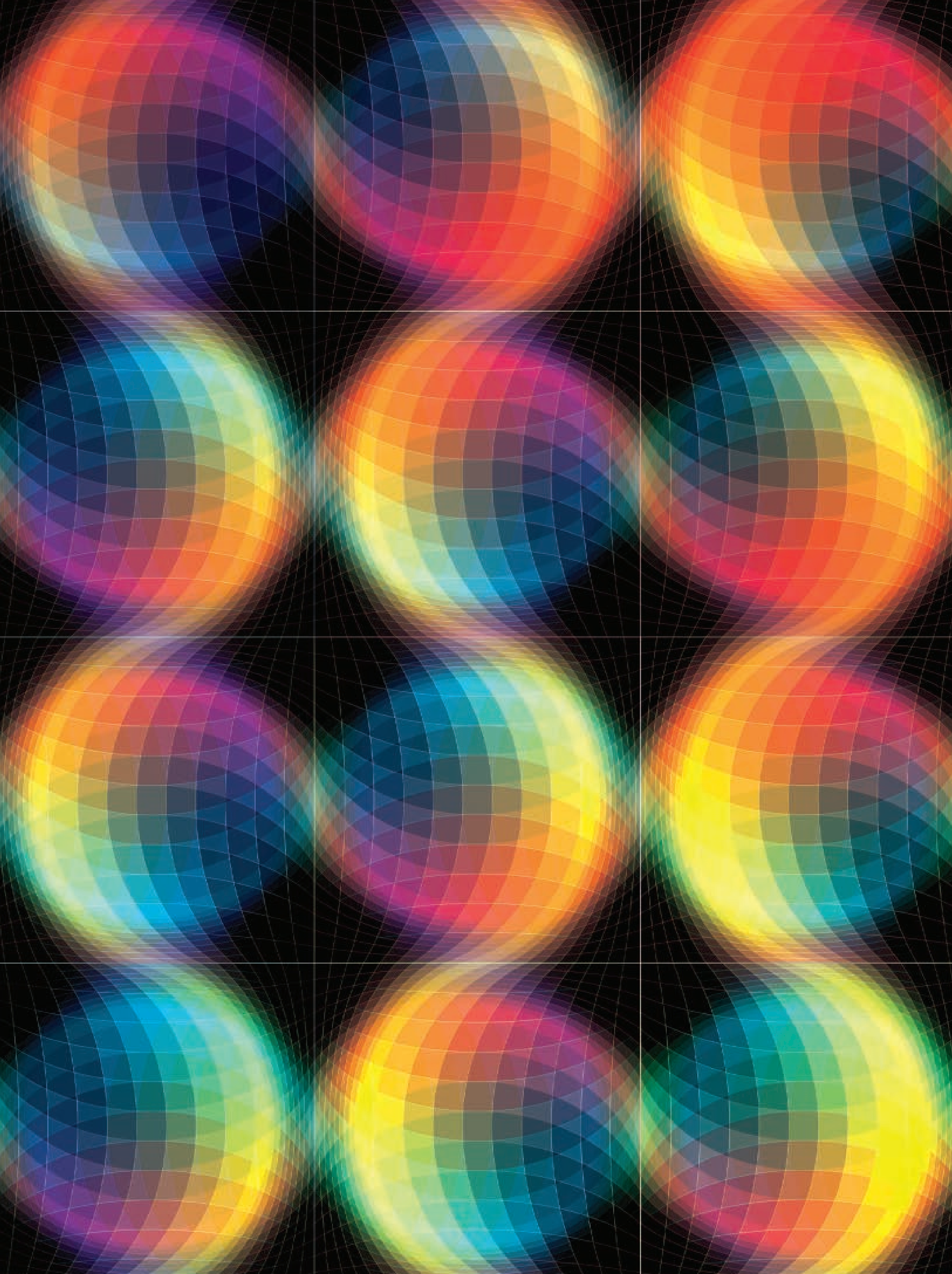
As shown in Figure 1A, every file of Erlang code is a module. Declarations within the file name the module (which must match the filename) and declare which functions can be called from other modules. Comments run from the percent sign (%) to the end of the line.

Factorial is implemented by two functions. Both are named `factorial`, but they have different numbers of arguments; hence, they are distinct. The definition of `factorial/2` (the

a See Erlang Web site www.erlang.org.

b See Erlang course www.erlang.org/download/armstrong_thesis_2003.pdf.

ILLUSTRATION BY ANDY GILMORE



two-argument version) is split into two clauses, separated by a semicolon. When `factorial/2` is called, the actual parameters are tested against the patterns in each clause head in turn to find the first match, then the body (after the arrow) is evaluated. The value of the final expression in the body is the return value of the call; no explicit return statement is needed. Erlang is dynamically typed, so a call to `factorial("pancake")` will compile but will raise a runtime exception when it fails to match any clause. Tail-calls are optimized, so this code will run in constant space.

Lists are enclosed in square brackets (see Figure 1B). A single vertical bar separates the first element from the rest of the list. If a list is used in a clause head pattern, it will match list values, separating them into their components. A list with a double vertical bar is a "list comprehension," constructing a list through generator and filter expressions. A double-plus (++) concatenates lists.

Tuples (vectors) are enclosed in curly braces (see Figure 1C). Tuples in patterns will extract components out of tuples that they match. Identifiers that start with an uppercase letter are variables; those that start in lowercase are *atoms* (symbolic constants such as enum values, but with no need to define a numerical representation). Boolean values are represented simply as atoms `true` and `false`. An underscore (_) in a pattern matches any value and does not create a binding. If the same fresh variable occurs several times in a pattern, the occurrences must be equal to match. Variables in Erlang are *single-assignment* (once a variable is bound to a value, that value never changes).

Not all list-processing operations can be expressed in list comprehensions. When we do need to write list-processing code directly, a common idiom is to provide one clause for handling the empty list and another for processing the first element of a non-empty list. The foldl/3 function shown in Figure 1D is a common utility that chains a two-argument function across a list, seeded by an initial value. Erlang allows anonymous functions ("fun's" or closures) to be defined on the fly, passed as arguments, or returned from functions.

Figure 1: Example1.erl.

A
```
-module(example1).
-export([factorial/1, qsort/1, member/2, foldl/3, sum/1]).

% Compute the factorial of a positive integer.
factorial(N) when is_integer(N), N > 0 -> factorial(N, 1).

% A helper function which maintains an accumulator.
factorial(1, Acc) -> Acc;
factorial(N, Acc) when N > 1 -> factorial(N - 1, N * Acc).
```

B
```
% Return a sorted copy of a list.
qsort([]) -> [];
qsort([Pivot | Xs]) ->
   qsort([X || X <- Xs, X < Pivot])
      ++ [Pivot]
      ++ qsort([X || X <- Xs, X >= Pivot]).
```

C
```
% Is X an element of a binary search tree?
member(_, empty) -> false;
member(X, {_, X, _}) -> true;
member(X, {Left, Y, _}) when X < Y -> member(X, Left);
member(X, {_, _, Right}) -> member(X, Right).
```

D
```
% "Fold" a function across elements of a list, seeding
% with an initial value.
% e.g. foldl(F, A0, [A, B, C]) = F(C, F(B, F(A, A0)))
foldl(_, Acc, []) -> Acc;
foldl(F, Acc, [X | Xs]) ->
   NewAcc = F(X, Acc),
   foldl(F, NewAcc, Xs).

% Give the sum of a list of numbers.
sum(Numbers) -> foldl(fun(N, Total) -> N + Total end, 0, Numbers).
```

Erlang has expressions that look like assignments but have a different semantics. The right-hand side of = is evaluated and then matched against the pattern on the left-hand side, just as when selecting a clause to match a function call. A new variable in a pattern will match against the corresponding value from the right-hand side.

Concurrent Erlang

Let's introduce concurrent Erlang by translating a small example from Java:

Sequence.java
```
// A shared counter.
public class Sequence {
     private int nextVal = 0;
     // Retrieve counter and
     // increment.
     public synchronized int
        getNext() {
           return nextVal++;
     }

// Re-initialize counter to zero.
     public synchronized void
     reset() {
        nextVal = 0;
     }
}
```

A sequence is created as an object on the heap, potentially accessible by multiple threads. The `synchronized` keyword means that all threads calling the method must first take a lock on the object. Under the protection of the lock, the shared state is read and updated, returning the pre-increment value. Without this synchronization, two threads could obtain the same value from `getNext()`, or the effects of a `reset()` could be ignored.

Let's start with a "raw" approach to Erlang, using the concurrency primitives directly.

sequence1.erl (raw implementation)

```
-module(sequence1).
-export([make_sequence/0,
  get_next/1, reset/1]).

% Create a new shared counter.
make_sequence() ->
  spawn(fun() ->
      sequence_loop(0)
  end).

sequence_loop(N) ->
  receive
      {From, get_next} ->
         From ! {self(), N},
         sequence_loop(N + 1);
      reset ->
         sequence_loop(0)
  end.

% Retrieve and increment.
get_next(Sequence) ->
  Sequence!{self(),get_next},
  receive
      {Sequence, N} -> N
  end.

% Re-initialize counter to zero.
reset(Sequence) ->
  Sequence ! reset.
```

The spawn/1 primitive creates a new *process*, returning its *process identifier* (pid) to the caller. An Erlang process, like a thread, is an independently scheduled sequential activity with its own call stack, but like an operating-system process, it shares no data with other processes—processes interact only by sending messages to each other. The self/0 primitive returns the pid of the caller. A pid is used to address messages to a process. Here the pid is also the data abstraction—a sequence is just the pid of a server process that understands our sequence-specific messaging protocol.

The new process starts executing the function specified in spawn/1 and will terminate when that function returns. Long-lived processes therefore avoid premature returns, often by executing a loop function. Tail-call optimization ensures that the stack does not grow in functions such as sequence_loop/1. The state of the sequence process is carried in the argument to this eternal call.

Messages are sent with the syntax *pid ! message*. A message can be any Erlang value, and it is sent atomically and immutably. The message is sent to the receiving process's mailbox, and the sender continues to execute—it does not wait for the receiving process to retrieve the message.

A process uses the receive expression to extract messages from its mailbox. It specifies a set of patterns and associated handler code and scans the mailbox looking for the first message that matches any of the patterns, blocking if no such message is found. This is the only blocking primitive in Erlang. Like the patterns in function clauses, the patterns in receive options match structures and bind new variables. If a pattern uses a variable that has already been bound to a value, then matching the pattern requires a match with that value, as in the value for Sequence in the receive expression in get_next/1.

The code here implements a simple client-server protocol. In a *call*, the client process sends a request message to the server process and blocks waiting for a response message. Here, the get_next/1 call request message is a two-element tuple: the client's own pid followed by the atom get_next. The client sends its own pid to let the server know where to send the response, and the get_next atom will let us differentiate this protocol operation from others. The server responds with its own two-element tuple: the server pid followed by the retrieved counter value. Including the server pid lets the client distinguish this response from other messages that might be sitting it its mailbox.

A *cast* is a request to a server that needs no response, so the protocol is just a request message. The reset/1 cast has a request message of just a bare atom.

Abstracting Protocols

Brief as it is, the Erlang implementation of sequences is much longer and less clear than the original Java version. Much of the code is not particular to sequences, however, so it should

Figure 2: server.erl.

```
-module(server).
-export([start/1, loop/2, call/2, cast/2]).
% Client-server messaging framework.
%
% The callback module implements the following callbacks:
% init() -> InitialState
% handle_call(Params, State) -> {Reply, NewState}
% handle_cast(Params, State) -> NewState

% Return the pid of a new server with the given callback module.
start(Module) ->
   spawn(fun() -> loop(Module, Module:init()) end).

loop(Module, State) ->
   receive
      {call, {Client, Id}, Params} ->
         {Reply, NewState} = Module:handle_call(Params, State),
         Client ! {Id, Reply},
         loop(Module, NewState);
      {cast, Params} ->
         NewState = Module:handle_cast(Params, State),
         loop(Module, NewState)
   end.

% Client-side function to call the server and return its reply.
call(Server, Params) ->
   Id = make_ref(),
   Server ! {call, {self(), Id}, Params},
   receive
      {Id, Reply} -> Reply
   end.

% Like call, but no reply is returned.
cast(Server, Params) ->
   Server ! {cast, Params}.
```

Figure 3: sequence2.erl (callback implementation).

```
-module(sequence2).
-export([make_sequence/0, get_next/1, reset/1]).
-export([init/0, handle_call/2, handle_cast/2]).
-export([test/0]).

% API
make_sequence()           -> server:start(sequence2).
get_next(Sequence)        -> server:call(Sequence, get_next).
reset(Sequence)           -> server:cast(Sequence, reset).

% Server callbacks
init()                    -> 0.
handle_call(get_next, N)  -> {N, N + 1}.
handle_cast(reset, _)     -> 0.

% Unit test: Return 'ok' or throw an exception.
test() ->
   0 = init(),
   {6, 7} = handle_call(get_next, 6),
   0 = handle_cast(reset, 101),
   ok.
```

Figure 4: Parallel call implementations.

A

```
% Make a set of server calls in parallel and return a
% list of their corresponding results.
% Calls is a list of {Server, Params} tuples.
multicall1(Calls) ->
   Ids = [send_call(Call) || Call <- Calls],
   collect_replies(Ids).

% Send a server call request message.
send_call({Server, Params}) ->
   Id = make_ref(),
   Server ! {call, {self(), Id}, Params},
   Id.

% Collect all replies in order.
collect_replies(Ids) ->
   [receive {Id, Result} -> Result end || Id <- Ids].
```

B

```
multicall2(Calls) ->
   Parent = self(),
   Pids = [worker(Parent, Call) || Call <- Calls],
   wait_all(Pids).

worker(Parent, {Server, Params}) ->
   spawn(fun() -> % create a worker process
      Result = server:call(Server, Params),
      Parent ! {self(), Result}
   end).

wait_all(Pids) ->
   [receive {Pid, Result} -> Result end || Pid <- Pids].
```

be possible to extract the message-passing machinery common to all client-server protocols into a common library.

Since we want to make the protocol independent of the specifics of sequences, we need to change it slightly. First, we distinguish client call requests from cast requests by tagging each sort of request message explicitly. Second, we strengthen the association of the request and response by tagging them with a per-call unique value. Armed with such a unique value, we use it instead of the server pid to distinguish the reply.

As shown in Figure 2, the `server` module contains the same structure as the `sequence1` module with the sequence-specific pieces removed. The syntax *`Module`*`:`*`function`* calls *`function`* in a module specified at runtime by an atom. Unique identifiers are generated by the `make_ref/0` primitive. It returns a new *reference*, which is a value guaranteed to be distinct from all other values in the program.

The server side of sequences is now boiled down to three one-line functions, as shown in Figure 3. Moreover, they are purely sequential, functional, and *deterministic* without message passing. This makes writing, analyzing, testing, and debugging much easier, so some sample unit tests are thrown in.

Standard Behaviours

Erlang's abstraction of a protocol pattern is called a *behaviour*. (We use the Commonwealth spelling as used in Erlang's source-code annotations.) A behaviour consists of a library that implements a common pattern of communication, plus the expected signatures of the callback functions. An instance of a behaviour needs some interface code wrapping the calls to the library plus the implementation callbacks, all largely free of message passing.

Such segregation of code improves robustness. When the callback functions avoid message-passing primitives, they become deterministic and frequently exhibit simple static types. By contrast, the behaviour library code is nondeterministic and challenges static type analysis. The behaviours are usually well tested and part of the standard library, however, leaving the application programmer the easier task of just coding the callbacks.

Callbacks have a purely functional interface. Information about any triggering message and current behaviour state are given as arguments, and outgoing messages and a new state are given in the return value. The process's "eternally looping function" is implemented in the library. This allows for simple unit testing of the callback functions.

Large Erlang applications make

heavy use of behaviours—direct use of the raw message-sending or receiving expressions is uncommon. In the Ericsson AXD301 telecom switch—the largest known Erlang project, with more than a million lines of code—nearly all the application code uses standard behaviours, a majority of which are the server behaviour.[1]

Erlang's OTP standard library provides three main behaviours:

Generic server (gen_server). The generic server is the most common behaviour. It abstracts the standard request-response message pattern used in client-server or remote procedure call protocols in distributed computing. It provides sophisticated functionality beyond our simple server module:

- Responses can be delayed by the server or delegated to another process.
- Calls have optional timeouts.
- The client monitors the server so that it receives immediate notification of a server failure instead of waiting for a timeout.

Generic finite state machine (gen_fsm). Many concurrent algorithms are specified in terms of a finite state machine model. The OTP library provides a convenient behaviour for this pattern. The message protocol that it obeys provides for clients to signal events to the state machine, possibly waiting for a synchronous reply. The application-specific callbacks handle these events, receiving the current state and passing a new state as a return value.

Generic event handler (gen_event). An event manager is a process that receives events as incoming messages, then dispatches those events to an arbitrary number of event handlers, each of which has its own module of callback functions and its own private state. Handlers can be dynamically added, changed, and deleted. Event handlers run application code for events, frequently selecting a subset to take action upon and ignoring the rest. This behaviour naturally models logging, monitoring, and "pubsub" systems. The OTP library provides off-the-shelf event handlers for spooling events to files or to a remote process or host.

Large Erlang applications make heavy use of behaviours—direct use of the raw message-sending or receiving expressions is uncommon.

The behaviour libraries provide functionality for dynamic debugging of a running program. They can be requested to display the current behaviour state, produce traces of messages received and sent, and provide statistics. Having this functionality automatically available to all applications gives Erlang programmers a profound advantage in delivering production-quality systems.

Worker Processes

Erlang applications can implement most of their functionality using long-lived processes that naturally fit a standard behaviour. Many applications, however, also need to create concurrent activities on the fly, often following a more ad-hoc protocol too unusual or trivial to be captured in the standard libraries.

Suppose we have a client that wants to make multiple server calls in parallel. One approach is to send the server protocol messages directly, shown in Figure 4A. The client sends well-formed server call messages to all servers, then collects their replies. The replies may arrive in the inbox in any order, but `collect_replies/1` will gather them in the order of the original list. The client may block waiting for the next reply even though other replies may be waiting. This doesn't slow things down, however, since the speed of the overall operation is determined by the slowest call.

To reimplement the protocol, we had to break the abstraction that the `server` behaviour offered. While this was simple for our toy example, the production-quality generic server in the Erlang standard library is far more involved. The setup for monitoring the server processes and the calculations for timeout management would make this code run on for several pages, and it would need to be rewritten if new features were added to the standard library.

Instead, we can reuse the existing behaviour code entirely by using *worker processes*—short-lived, special-purpose processes that don't execute a standard behaviour. Using worker processes, this code becomes that shown in Figure 4B.

We spawn a new worker process for each call. Each makes the requested call and then replies to the parent, using its own pid as a tag. The parent then receives each reply in turn, gathering them in a list. The client-side code for a server call is reused entirely as is.

By using worker processes, libraries are free to use receive expressions as needed without worrying about blocking their caller. If the caller does not wish to block, it is always free to spawn a worker.

Dangers of Concurrency

Though it eliminates shared state, Erlang is not immune to races. The server behaviour allows its application code to execute as a critical section accessing protected data, but it's always possible to draw the lines of this protection incorrectly.

Figure 5, for example, illustrates that if we had implemented sequences with raw primitives to read and write the counter, we would be just as vulnerable to races as a shared-state implementation that forgot to take locks.

This code is insidious as it will pass simple unit tests and can perform reliably in the field for a long time before it silently encounters an error. Both the client-side wrappers and server-side call-backs, however, look quite different from those of the correct implementation. By contrast, an incorrect shared-state program would look nearly identical to a correct one. It takes a trained eye to inspect a shared-state program and notice the missing lock requests.

All standard errors in concurrent programming have their equivalents in Erlang: races, deadlock, livelock, starvation, and so on. Even with the help Erlang provides, concurrent programming is far from easy, and the nondeterminism of concurrency means that it is always difficult to know when the last bug has been removed.

Testing helps eliminate most gross errors—to the extent that the test cases model the behaviour encountered in the field. Injecting timing jitter and allowing long burn-in times will help the coverage; the combinatorial explosion of possible event orderings in a concurrent system means that no nontrivial application can be tested for all possible cases.

When reasonable efforts at testing reach their end, the remaining bugs are usually heisenbugs,[5] which occur nondeterministically but rarely. They can be seen only when some unusual timing pattern emerges in execution. They are the bane of debugging since they are difficult to reproduce, but this curse is also a blessing in disguise. If a heisenbug is difficult to reproduce, then if you rerun the computation, you might not see the bug. This suggests that flaws in concurrent programs, while unavoidable, can have their impact lessened with an automatic retry mechanism—as long as the impact of the initial bug event can be detected and constrained.

Figure 5: badsequence.erl.

```
% BAD - race-prone implementation - do not use - BAD
-module(badsequence).
-export([make_sequence/0, get_next/1, reset/1]).
-export([init/0, handle_call/2, handle_cast/2]).

% API
make_sequence() -> server:start(badsequence).
get_next(Sequence) ->
   N = read(Sequence),
   write(Sequence, N + 1), % BAD: race!
   N.
reset(Sequence) -> write(Sequence, 0).
read(Sequence) -> server:call (Sequence, read).
write(Sequence, N) ->
   server:cast(Sequence, {write, N}).

% Server callbacks
init()                          -> 0.
handle_call(read, N)            -> {N, N}.
handle_cast({write, N}, _)      -> N.
```

Failure and Supervision

Erlang is a safe language—all runtime faults, such as division by zero, an out-of-range index, or sending a message to a process that has terminated, result in clearly defined behavior, usually an exception. Application code can install exception handlers to contain and recover from expected faults, but an uncaught exception means that the process cannot continue to run. Such a process is said to have failed.

Sometimes a process can get stuck in an infinite loop instead of failing overtly. We can guard against stuck processes with internal watchdog processes. These watchdogs make periodic calls to various corners of the running application, ideally causing a chain of events that cover all long-lived processes, and fail if they don't receive a response within a generous but finite timeout. Process failure is the uniform way of detecting errors in Erlang.

Erlang's error-handling philosophy stems from the observation that any robust cluster of hardware must consist of at least two machines, one of which can react to the failure of the other and take steps toward recovery.[2] If the recovery mechanism were on the broken machine, it would be broken, too. The recovery mechanism must be outside the range of the failure. In Erlang, the process is not only the unit of concurrency, but also the range of failure. Since processes share no state, a fatal error in a process makes its state unavailable but won't corrupt the state of other processes.

Erlang provides two primitives for one process to notice the failure of another. Establishing *monitoring* of another process creates a one-way notification of failure, and *linking* two processes establishes mutual notification. Monitoring is used during temporary relationships, such as a client-server call, and mutual linking is used for more permanent relationships. By default, when a fault notification is delivered to a linked process, it causes the receiver to fail as well, but a process-local flag can be set to turn fault notification into an ordinary message that can be handled by a `receive` expression.

In general application programming, robust server deployments include an external "nanny" that will monitor the running operating system process and restart it if it fails. The restarted process reinitializes itself by reading its persistent state from disk and then resumes running. Any pending operations and volatile state will be lost, but assuming that the persistent state isn't irreparably corrupted, the service can resume.

The Erlang version of a nanny is the *supervisor* behaviour. A supervisor process spawns a set of child processes and links to them so it will be informed if they fail. A supervisor uses an ini-tialization callback to specify a strategy and a list of child specifications. A child specification gives instructions on how to launch a new child. The strategy tells the supervisor what to do if one of its children dies: restart that child, restart all children, or several other possibilities. If the child died from a persistent condition rather than a bad command or a rare heisenbug, then the restarted child will just fail again. To avoid looping forever, the supervisor's strategy also gives a maximum rate of restarting. If restarts exceed this rate, the supervisor itself will fail.

Children can be normal behaviour-running processes, or they can be supervisors themselves, giving rise to a tree structure of supervision. If a restart fails to clear an error, then it will trigger a supervisor subtree failure, resulting in a restart with an even wider scope. At the root of the supervision tree, an application can choose the overall strategy, such as retrying forever, quitting, or possibly restarting the Erlang virtual machine.

Since linkage is bidirectional, a failing server will notify or fail the children under it. Ephemeral worker processes are usually spawned linked to their long-lived parent. If the parent fails, the workers automatically fail, too. This linking prevents uncollected workers from accumulating in the system. In a properly written Erlang application, all processes are linked into the supervision tree so that a top-level supervision restart can clean up all running processes.

In this way, a concurrent Erlang application vulnerable to occasional deadlocks, starvations, or infinite loops can still work robustly in the field unattended.

With the increasing importance of concurrent programming, Erlang is seeing growing interest and adoption. Indeed, Erlang is branded as a "concurrency-oriented" language.

Implementation, Performance, and Scalability

Erlang's concurrency is built upon the simple primitives of process spawning and message passing, and its programming style is built on the assumption that these primitives have a low overhead. The number of processes must scale as well—imagine how constrained object-oriented programming would be if there could be no more than a few hundred objects in the system.

For Erlang to be portable, it cannot assume that its host operating system has fast interprocess communication and context switching or allows a truly scalable number of schedulable activities in the kernel. Therefore, the Erlang *emulator* (virtual machine) takes care of scheduling, memory management, and message passing at the user level.

An Erlang instance is a single operating-system process with multiple operating-system threads executing in it, possibly scheduled across multiple processors or cores. These threads execute a user-level scheduler to run Erlang processes. A scheduled process will run until it blocks or until its time slice runs out. Since the process is running Erlang code, the emulator can arrange for the scheduling slice to end at a time when the process context is minimal, minimizing the context switch time.

Each process has a small, dedicated memory area for its heap and stack. A two-generation copying collector reclaims storage, and the memory area may grow over time. The size starts small—a few hundred machine words—but can grow to gigabytes. The Erlang process stack is separate from the C runtime stack in the emulator and has no minimal size or required granularity. This lets processes be lightweight.

By default, the Erlang emulator interprets the intermediate code produced by the compiler. Many substantial Erlang programs can run sufficiently fast without using the native-code compiler. This is because Erlang is a high-level language and deals with large, abstract objects. When running, even the interpreter spends most of its time executing within the highly tuned runtime system written in C. For example, when copying bulk data between network sockets, interpreted Erlang performs on par with a custom C program to do the same task.[4]

The significant test of the implementation's efficiency is the practicality of the worker process idiom, as demonstrated by the `multicall2` code shown earlier. Spawning worker processes would seem to be much less efficient than sending messages directly. Not only does the parent have to spawn and destroy a process, but also the worker needs extra message hops to return the results. In most programming environments, these overheads would be prohibitive, but in Erlang, the concurrency primitives (including process spawning) are lightweight enough that the overhead is usually negligible.

Not only do worker processes have negligible overhead, but they also increase efficiency in many cases. When a process exits, all of its memory can be immediately reclaimed. A short-lived process might not even need a collection cycle. Per-process heaps also eliminate global collection pauses, achieving soft real-time levels of latency. For this reason, Erlang programmers avoid reusable pools of processes and instead create new processes when needed and discard them afterward.

Since values in Erlang are immutable, it's up to the implementation whether the message is copied when sent or whether it is sent by reference. Copying would seem to be the slower option in all situations, but sending messages by reference requires coordination of garbage collection between processes: either a shared heap space or maintenance of interregion links. For many applications, the overhead of copying is small compared with the benefit from short collection times and fast reclamation of space from ephemeral processes. The low penalty for copying is driven by an important exception in send-by-copy: raw binary data is always sent by reference, which doesn't complicate garbage collection since the raw binary data cannot contain pointers to other structures.

The Erlang emulator can create a new Erlang process in less than a microsecond and run millions of processes simultaneously. Each process takes less than a kilobyte of space. Message passing and context switching take hundreds of nanoseconds.

Because of its performance characteristics and language and library support, Erlang is particularly good for:

- Irregular concurrency—applications that need to derive parallelism from disparate concurrent tasks
- Network servers
- Distributed systems
- Parallel databases
- GUIs and other interactive programs
- Monitoring, control, and testing tools

So when is Erlang not an appropriate programming language, for efficiency or other reasons? Erlang tends not to be good for:

- Concurrency more appropriate to synchronized parallel execution
- Floating-point-intensive code
- Code requiring nonportable instructions
- Code requiring an aggressive compiler (Erlang entries in language benchmark shoot-outs are unimpressive—except for process spawning and message passing)
- Projects to implement libraries that must run under other execution environments, such as JVM (Java Virtual Machine) or CLR (Common Language Runtime)
- Projects that require the use of extensive libraries written in other languages

Erlang can still form part of a larger solution in combination with other languages, however. At a minimum, Erlang programs can speak text or binary protocols over standard interprocess communication mechanisms. In addition, Erlang provides a C library that other applications can link with that will allow them to send and receive Erlang messages and be monitored by an Erlang controlling program, appearing to it as just another (Erlang) process.

Conclusion

With the increasing importance of concurrent programming, Erlang is seeing growing interest and adoption. Indeed, Erlang is branded as a "concurrency-oriented" language. The standard Erlang distribution is under active development. Many high-quality libraries and applications are freely available for:

- Network services
- GUIs for 3D modeling
- Batch typesetting
- Telecom protocol stacks
- Electronic payment systems
- HTML and XML generation and parsing
- Database implementations and ODBC (Open Database Connectivity) bindings

Several companies offer commercial products and services implemented in Erlang for telecom, electronic payment systems, and social networking chat. Erlang-based Web servers are notable for their high performance and scalability.[3]

Concurrent programming will never be easy, but with Erlang, developers have a chance to use a language built from the ground up for the task and with incredible resilience engineered in its runtime system and standard libraries.

The standard Erlang implementation and its documentation, ported to Unix and Microsoft Windows platforms, is open source and available for free download from http://erlang.org. You can find a community forum at http://trapexit.org, which also mirrors several mailing lists. C

The author would like to thank Romain Lenglet and JJ Furman for their feedback.

References

1. Armstrong, J. Making reliable distributed systems in the presence of software errors. Ph.D. thesis (2003). Swedish Institute of Computer Science; www.erlang.org/download/armstrong_thesis_2003.pdf.
2. Armstrong, J. *Programming Erlang: Software for a Concurrent World.* The Pragmatic Bookshelf, Raleigh, NC, 2007.
3. Brown, B. Application server performance testing, includes Django, ErlyWeb, Rails, and others; http://berlinbrowndev.blogspot.com/2008/08/application-server-benchmarks-including.html.
4. Lystig Fritchie, S., Larson, J., Christenson, N., Jones, D., and Ohman, L. Sendmail meets Erlang: Experiences using Erlang for email applications. In *Proceedings of the Erlang User's Conference*, 2000; www.erlang.se/euc/00/euc00-sendmail.ps.
5. Steele, G.L. and Raymond, E.S. *The New Hacker's Dictionary*, 3rd edition. MIT Press, Cambridge, MA, 1996.

Jim Larson (jimlarson@google.com) is a software engineer at Google, implementing large-scale distributed storage systems. He has worked with Erlang for commercial products since 1999 and is the architect of the replication engine of the Amazon SimpleDB Web service at Amazon.com.

A previous version of this article appeared in the September 2008 issue of *ACM Queue*.

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 **full 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 AUTHORIZED PARTNER

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 **full 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.

Safari® Books Online

500 Online Books from Books24x7

All Professional and Student Members also have **full 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.

books24x7® *a SkillSoft Company*

acm Association for Computing Machinery

Advancing Computing as a Science & Profession

pd.acm.org

www.acm.org/join

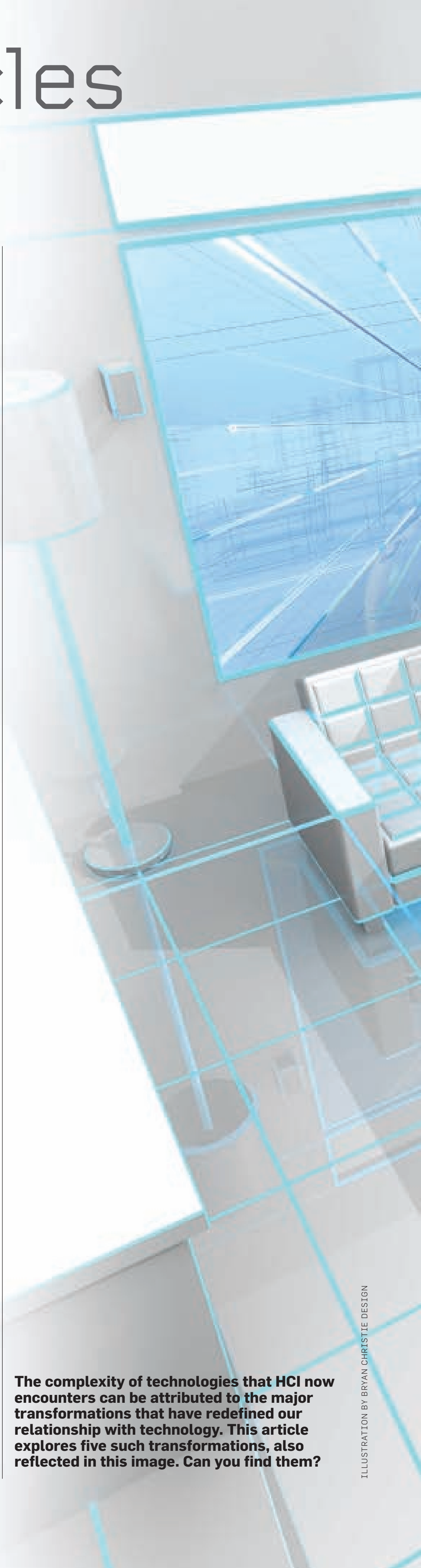
DOI:10.1145/1467247.1467265

HCI experts must broaden the field's scope and adopt new methods to be useful in 21st-century sociotechnical environments.

BY ABIGAIL SELLEN, YVONNE ROGERS, RICHARD HARPER, AND TOM RODDEN

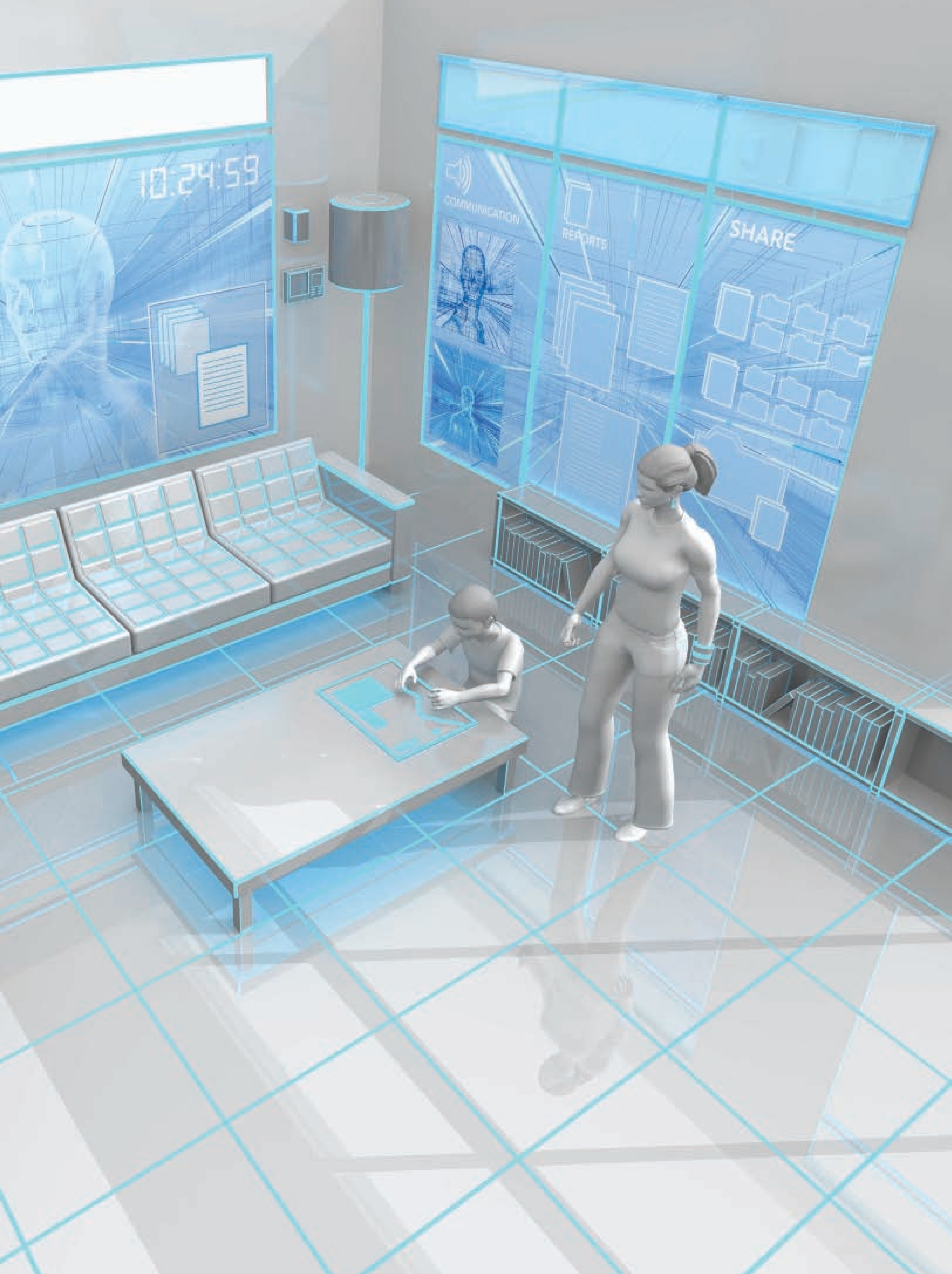
Reflecting Human Values in the Digital Age

THE FIELD OF human-computer interaction (HCI) came into being more than 25 years ago with the mission of understanding the relationship between humans and computers, often with an eye toward improving the technology's design. But that relationship has since been altered so radically—changes in the sociotechnical landscape have been so great—that many in the community of HCI researchers and practitioners are questioning where the field is headed. Computer systems now intrude on our lives as well as disappear into the world around us, they monitor as well as guide us, and they coerce as well as aid us. Thus there are debates about such fundamentals as what HCI's goals should be, how it should do its work, and whether its methods remain relevant.



The complexity of technologies that HCI now encounters can be attributed to the major transformations that have redefined our relationship with technology. This article explores five such transformations, also reflected in this image. Can you find them?

ILLUSTRATION BY BRYAN CHRISTIE DESIGN


10:24:59
COMMUNICATION
REPORTS
SHARE

In March 2007, academic and industrial researchers from many different countries and diverse backgrounds, including computing, social science, and design, met in Seville, Spain, for a two-day workshop entitled "HCI in 2020." The event, sponsored by Microsoft Research Cambridge, U.K., was a chance to air views, reflect, and discuss the future of HCI as well as issues of central importance to the field. Needless to say, participants expressed a wide range of opinions, but they were virtually unanimous that the field of HCI must change its scope and methods if it is to remain relevant in the 21st century.

While the researchers agreed as well on the need to keep human values at HCI's core, they highlighted the fact that our changing relationship with computers means that determining what these values might be and coming to understand them require greater finesse than ever before. If in the past HCI was in the business of understanding how people could become more efficient through the use of computers, the challenge confronting the field now is to deal with issues that are much more complex and subtle. Here we summarize these issues, basing our discussion on the workshop's report *Being Human: Human-Computer Interaction in the Year 2020*.[1]

Values are not something that can be catalogued like books in a library but are bound to each other in complex weaves that, when tugged in one place, pull values elsewhere out of shape.

A Brief Look Back

When the field of HCI was in its infancy, a common activity was to model a user's interaction with a desktop computer so that the interface between person and machine could be optimized. HCI was mainly a scientific and engineering endeavor, using techniques derived from cognitive psychology and human-factors engineering.[8] What went on "inside the head" of a user was specified by observing behavior under controlled conditions, inferring what kinds of perceptual, cognitive, and motor processes were involved, and developing pertinent theories.[2] Methods for optimizing "usability" were devised, and iterative testing with real users was seen as prerequisite to introducing any new software or hardware product.

During the 1990s, the objectives of HCI began changing along with the growth of communication networks that link computers. Researchers started asking how users, with the aid of computers, might interact with each other.[13] Researchers with backgrounds in more socially oriented sciences, such as sociology and anthropology, began to engage with HCI. These disciplines emphasized not only the effects of computing on *groups* of users but also how those very same groups appropriated computers, interpreted them, and socially and emotionally experienced their relationships with the technology. Several of the approaches of these disciplines were added to the mix with ethnographic approaches being especially visible.

The practical result of these developments is that HCI has become an academic discipline in its own right, with conferences dedicated to the subject as well as departments and courses offering HCI as a speciality, and it has also become an integral part of the design processes—typically, user-centered—for nearly all technology companies.[14] Moreover, an understanding of HCI (if not its details or techniques) has seeped into the broader consciousness, as the common use of terms such as "user-friendliness" and "user experience" in the news media and everyday conversation attest. Such awareness, among practitioners and users alike, has encompassed computers not only in the conventional sense of, say, desktop systems but also as they are manifested in cars, airplanes, mobile phones, and a broad array of other products.

In parallel, important changes in research objectives have also taken place within the field. The HCI of today is exploring diverse new areas beyond the workplace, including the role of technology in home life and education and even delving into such diverse areas as play, spirituality, and sexuality. HCI is now more multidisciplinary than ever, with a significant percentage of the community coming from the design world. This shift has caused the field's practitioners to think more broadly about their design goals, taking into account not just how technology might be functional or useful but also how it might provoke, engage, disturb, or delight.

Transformations in Interaction

Despite the progress, gradual but now

The growth in hyperconnectivity carries with it both the benefits and the pressures of being connected "anywhere, anytime."

very visible transformations in our relationship to computers are leading many in HCI—including participants in the Seville workshop—to urge a radical rethinking of the underpinnings of HCI: its mission, goals, and philosophical approach, both for research and practice. In essence, the claim is that the interaction between values and technology needs to be much more carefully navigated than before. This is not a simple choice between designing for what is desirable as opposed to what is reprehensible; HCI specialists also need to be astutely aware of how one set of design choices might highlight certain values at the expense of others. In other words, values are not something that can be catalogued like books in a library but are bound to each other in complex weaves that, when tugged in one place, pull values elsewhere out of shape. Further, now more than ever, the diversity, scope, and complexity of the technologies that HCI deals with make tradeoffs between values a conundrum, not a platitude.

The reasons for this new complexity can be attributed in large part to the major transformations that have redefined our relationship with technology. Here we characterize five such transformations, each of which continues to alter the ways in which humans coexist with computers, interact with them, decide what problems to focus on, and pursue solutions.

The first transformation—*the end of interface stability*—has to do with how computers can no longer be defined by reference to a single interface but rather by many different interfaces or, alternatively, none at all. For example, some computers encroach ever more deeply into our own personal spaces: we carry them, wear them, and may even have them implanted within us. Other forms of computers are disappearing into the richness and complexity of the world around us. They are increasingly embedded in everyday objects; not just toys, home appliances, and cars but also books, clothing, and furniture. And they are increasingly part of our environments, in public spaces such as airports, garages, and shopping malls as well as in the private spaces of homes and offices. In each case, where the interface might be, or even if there is an interface at all, is an open question. All of this has consequences for HCI. After all, the assumption that the locus of human-machine interaction is obvious (and hence can be observed, researched, and designed for) has been at the core of HCI since its foundation. If this is no longer the case, then what an interface might be, where it is, what it allows a user to do, and even whether there is one at all are now the issues that a future-looking HCI must address.

A second transformation, *the growth of techno-dependency*, refers to the fact that changes in how we live with and use technology have resulted in our becoming ever more reliant on it. There is of course no news in saying that society and individuals alike depend on a technological infrastructure. But what is different about this transformation is that computational dependence is more complex, fraught with more snag points, and vulnerable to more forms of attack. It is not simply that we are increasingly using computers in routine but selected activities, such as to write reports or do our tax returns. Computing now underpins almost every aspect of our lives, from shopping to travel, from work to medicine. At the same time, computers are becoming ever more sophisticated and autonomous. As a result, not only is our reliance on them growing but computers themselves are increasingly reliant on each other. The extent of our need for computers—characterized by a wide diversity of technologies, an "always-on" infrastructure, and an interconnected web of systems—creates new concerns, new design opportunities, and new research topics that specialists in HCI are obliged to address.

A third transformation is *the growth in hyperconnectivity*, the influential role of communication technologies in tying us together in ways that were unimaginable even as recently as 10 years ago. Despite the ability of such new tools to improve efficiency and save us time, such "digital presence" increasingly consumes our time rather than saves it. Communication devices are now filling our lives up instead of releasing us from burden. Yet hy-

The "interface" between humans and computers is harder than ever to define. We can interact with computers just by walking through a public space.

TOP: PHOTOGRAPH BY IAN FORRESTER, BOTTOM: PHOTOGRAPHY COURTESY OF PHILIPS DESIGN perconnectivity also has the power to mobilize us, as citizens and members of global communities; we are now in touch in more ways, and with more people, than ever. What these changes mean, how one designs for them, and how one judges value within the myriad forms of being in touch are all substantive issues for HCI.

Fourth, our heightened ability to be in touch is equalled by a passion to capture more and more information about people's lives and actions—information that hitherto would have been discarded or forgotten. This trend is reflecting as well as driving the massive gains in computer networks' capacity. What it means to record, why we record, and what we do with the collected material is changing hand-in-hand with the systems we use to capture, manage, share, and archive these burgeoning stores of personal data. Each of us is developing an ever-increasing "digital footprint"—sometimes in ways we desire, sometimes not, and often in ways we know little about—not only on a personal level but also within the databases of government agencies and other public, as well as private, institutions. We call this transformation *the end of the ephemeral*.

Finally, the proliferation of new kinds of digital tools (exemplified by Web 2.0) and their appropriation by people from all walks of life are enabling us to work, play, and express ourselves in new ways. Computers were once limited to the automation and mechanization of routine aspects of work or problem-solving. Now, more than ever, they are also instruments for creativity. This trend is manifested not only in the explosion of computer tools for play and self-expression; it also propels more "serious" pursuits. For example, computational tools are enabling advances in the world of science and medicine as they assist researchers in discerning, analyzing, and solving problems. This fifth transformation—*the growth of creative engagement*—underscores the fact that flexible computer tools, which can be assembled and appropriated in new ways, allows us to see the world in wholly new ways too. Computer-enabled creativity means we can all become our own producers, programmers, and publishers, whether in our personal or professional lives, with potentially far-reaching consequences.

New Questions for a Future-Looking HCI

The five transformations are provoking questions that HCI has not had to address before, as they concern issues that simply did not arise in a world where using a computer essentially meant a person sitting in front of a desktop machine doing email, writing a document, or working on a spreadsheet. Because our relationship with computing is now far more extensive and complex, these new questions deal with how we design for the emerging interaction paradigms.

For example, the end of interface stability raises questions such as:

- What interaction techniques are appropriate if devices embedded within us have no explicit or recognizable "interface?"
- Should new interaction techniques build on the skills we have already acquired for dealing with far less complicated systems? And if so, how?
- How do we enable people to understand the complexity of new ecosystems of technologies, and the results of interacting with them, so as to proceed most effectively?

Our growing dependency on computing provokes a different set of questions, including:

- How do we design computer systems to help people cope when infrastructures break down or when devices malfunction or are lost?
- What will be the taken-for-granted technologies of the future and how might they alter the skill sets of the people for whom we must design?
- With computers becoming increasingly autonomous, seemingly able to make their own decisions, what will be an appropriate style of human-computer interaction?

The end of the ephemeral leads us to consider what is being recorded, stored, and analyzed regarding our beliefs, preferences, and everyday

Questions of Broader Impact

Computers will soon be able to monitor the bodily functions of people without requiring their awareness or necessarily seeking their permission.

Who should have the right to access and control information from embedded devices? It is obvious that such devices will alter the knowledge that medical professionals will have of a patient's body, but less obvious is how this will alter their perception of the sanctity of the body. Similarly, the output of such devices will alter the conception that people have of themselves, but in what ways and to what end?

An increasingly complex set of computing devices will pervade our homes.

Who is responsible for preventing breakdowns, fixing problems, and ensuring protection from unplanned and undesirable consequences? Users or householders will need to be accountable to some extent, but in other cases it may need to be the service provider or government. In addition, the identity of the user can be difficult to ascertain when venturing beyond the work setting. At home, are children to be held responsible for the consequences of their interactions with technology? Or does responsibility rest on a child's parents or legal custodians?

New technologies will continue to shift the balance of labor between people and machines in ways that will change our skills, strengthening some and atrophying others.

The increased burdens taken on by machines may come at a cost, in terms of human skills, that is not so easy to see or understand. How do we examine and judge what is the best balance? Human factors engineers sought to answer this question for the workplace, but what about social systems or households, for example? How does one analyze the relationship between loss of engagement in one area and the opening up of opportunities elsewhere if the activities involved have to do with play rather than work, expressiveness rather than calculation, desire rather than labor?

Digital footprints are expanding in ways that we understand and are visible but also in ways that we don't comprehend or see.

As an example, we place tagged photos of ourselves on photo-sharing sites only to find images of ourselves already there. Should we have the right to remove such pictures? What about other kinds of stored information about ourselves? Do we want to have a copyright on our own digital footprints? If this applies to the digital world, what does it imply for the physical world?

actions—and interactions. Questions include:

▸ What computer technologies are needed to effectively manage vast quantities of personal data?

▸ How do people learn about their digital footprint as well as the tools that can help them interrogate the systems involved and analyze the data?

▸ How do we design computer systems so as to give people feedback about, and control over, information-capturing processes?

▸ How can the capture of information and the need for privacy be balanced through design?

Taken together, these and other transformation-related questions point to a very different kind of agenda, for researchers, practitioners, and technology designers alike, from the one that was appropriate for HCI in the 1980s and 1990s.

But in addition to new questions about interaction and design, many of the issues these transformations raise are much more far-reaching. They include how society should react to the changes that computer systems engender—how their impact will be dealt with in different situations, places, and cultures—and a range of moral concerns. The sidebar here—"Questions of Broader Impact"—posits some of these changes, followed by examples of the new kinds of ethical questions they raise.

Making judgments about new computer technologies, and how they will affect us and the social fabric of which we are a part, is not straightforward. Research methods must capture how the use of technologies may unfold over time and in different situations.

Human Values in the Face of Change

Should the HCI community be addressing these more far-reaching kinds of questions? And if so, is it equipped to take on the task? The participants at the Seville workshop agreed that it should—and also that a quite different mind-set is required.[1]

To begin with, researchers and practitioners in HCI need to analyze the wider set of issues that are now in play. Central to the new agenda is recognizing what it means to be human in a digital future. Human values, in all their diversity, should be charted in relation to how they are supported, augmented, or constrained by technological developments. In many ways, this is arguing for a strengthening of what has always been important to HCI: a focus on human-centered design, keeping firmly in mind what users—people—need and want from technology. The trouble is that the values that systems often impinge on are not the kind that can be easily inventoried. For instance, values related to technologies that capture our digital footprint may support our recollection of the past and influence ideas of selfhood just as much as they might imply more measurable ideals related to bureaucratic efficiency (for example, keeping good records). Computational technology affects both, though the audit of one is considerably more difficult than that of the other.

It follows that the field of HCI needs to extend its approach in order to encompass the often complex and diverse patterns of human interests and aspirations. This means that the methods of HCI, and the disciplines it engages with, will have to change.

Important steps have already been taken in this direction—in the concept of "use," for example. A growing number of researchers and practitioners have begun explicating the nature of use as a question of "experience" and how it unfolds over time. This has largely involved the definition of subjective qualities. Analysts have used concepts like pleasure, aesthetics, fun, and flow, on the one hand, and boredom, annoyance, and intrusiveness, on the other, to describe the multifaceted nature of "felt" experiences.[10] In addition, HCI specialists such as Norman[11] have modeled how we respond to technology at a visceral or emotional level as well as at a deliberate and reflective one. They have also described a more comprehensive life cycle of our response to technology, from when it first grabs our attention and entices us, through our ongoing relationship with that technology, and finally to when it is eclipsed by other technologies and we abandon it. These ways of conceptualizing users' experience have opened up many new possibilities for research and design.

An emphasis on the individual and the phenomenology of his or her experiences is a natural consequence of HCI's traditional starting point: the user. But it should be obvious that as HCI moves forward and seeks to address the changes cited previously, the user, however well understood, is only part of a larger system—or set of systems. Much effort also needs to be expended on determining what is desirable within a place, an institution, or a society. Values such as personal privacy, health, ownership, fair play, and security are obvious candidates for analysis, but so too are public, institutional, and civic identities. The values treasured by the individual are not always in harmony with those of institutions or the society; nor, on the other hand, are they always inimical to one another. Here specialists in HCI can learn a great deal from disciplines, such as sociology and anthropology, that focus on organizations and cultures. The bottom line is that the field of HCI needs to take into account the broader context within which human values are expressed.

Some HCI researchers are indeed beginning to emphasize human values as central to research and design,[3,5,6,13] while others have been attempting to define a "third paradigm"[9] that draws on ideas of embodiment[4] such as, taking into account the interactions and conversations that happen in our physical and social worlds that provide meaning. These alternative approaches stress that a deep understanding of our interactions with technology cannot be divorced from their contexts. The meaning of technology is created within specific situations, and not just by individuals but often by many stakeholders.

Yet making judgments about new computer technologies, and how they will affect us and the social fabric of which we are a part, is not straightforward. Research methods must capture how the use of technologies may unfold over time and in different situations. Consider that computers can help connect us to others, but by the same token it is important that they sometimes allow us to be isolated. Likewise, computers can support our industriousness but at other times we may want to "switch off."

Moreover, such choices are not always ours alone to make; it is not simply users and their own particular aspirations that are involved. For example, workplaces reserve the right to summon their staff to be industrious. In other words, sometimes communications are meant to be heard even when the audience does not especially want to listen. As Peters notes in *Speaking into the Air*,[12] communications can be about communion as well as about information exchange. So design tradeoffs need to be considered not just in terms of our local interaction with a technology but also in terms of weighing the various moral, personal, and institutional consequences.

In a world where people's movements and transactions can be tracked—where individuals trigger nondeliberate events just by being in a certain place, physical or virtual, at a certain time—the notion of interaction itself is being fundamentally altered.

A New Approach for HCI

We propose, then, that a broader approach is needed for tackling the new kinds of questions that the transformations are raising. But what are the practical implications of such an avenue? What does it mean for the field of HCI?

Folding human values into the research and design cycle. Our first suggestion, described more fully in the Seville workshop's *Being Human* report, is to extend the ways in which user-centered research and design are conducted by explicitly addressing human values.

A simplified but helpful model of current practice is that projects typically follow an iterative cycle, comprised of four fundamental stages, in which HCI specialists sequentially *study*, *design*, *build*, and *evaluate* technology with users. The goal, for example, may be to design a particular computing technology in order to improve upon a given experience. Initial research involves finding out about people's current practices, for which ethnographic studies, logging of user interactions, and surveys are commonly employed. Based on the information gathered, the specialists begin to focus on the why, what, and how of designing something better. To aid in the process, usability and user-experience goals are identified and conceptual models developed. Prototypes are built, evaluated, and iterated on until it is determined whether the new technology can meet the user goals and whether the new user experience is judged by the target group to be valuable and enjoyable.

The *Being Human* report proposes that a new agenda for HCI should enhance this model by adding another stage—an initial stage, called *understand*—which aims to pinpoint the human values that the technology

The "History Tablecloth," developed by the Interaction Research Studio (Goldsmith's College, University of London), is an example of embedding computing in everyday objects. When items are left on the cloth it begins to glow beneath them, creating a slowly expanding halo. When the items are removed, the glow gradually fades.

in question will be designed to serve. Depending on the values of interest, this analysis might need to draw on disciplines as diverse as philosophy, psychology, art, sociology, cultural studies, and architecture, for example. It might also mean collaborating with the stakeholders behind the technology to ascertain what kinds of enduring values they expect their users to derive from the product.

Consider, for example, that there might be an interest in developing new interactive tabletop applications for working with digital photos. The *understand* stage of the work would involve clarifying what kinds of human values might be made possible through such interactions. Is it about supporting social connectivity around photographs? About play and creativity with digital images? About archiving photographs and other materials in order to preserve and honor family history? Or is it about allowing individuals to reflect on their personal past through images? The list could go on.

Ultimately, this stage is about making basic choices. It requires specifying up front the kinds of users targeted, and in which domains of activity, environments, or cultures. In other words, the stage involves choosing the values being designed for. Its investigations will then point to some fundamental research that needs to be conducted, relevant research that has already been carried out, or some combination of the two. The stage may equally well involve experts from diverse disciplines, such as social historians, game designers, or specialists in the psychology of memory, to cite but a few.

Further, the extended approach to HCI is intended to enable human values to be folded into the mix not just at the *understand* stage but the other four stages as well. In the report, we give fuller examples of how choices made about the human values of interest can provide guidance in the *study*, *design*, *build*, and *evaluate* phases. Key here is that the analysis should not just take into account people's interactions with computer technology but also with the environment, with everyday objects, with other human beings, and with the changing landscape that the "new tech" brings to their world.

Forming new partnerships. Aside from changes in methodology, HCI also needs to develop partnerships with other disciplines that traditionally have not been part of the field. One reason has been outlined here—that different human values, as expressed in diverse contexts, point to the need for all kinds of expertise to deeply understand and creatively design for the relationships between those values and technology.

But other reasons have to do with questions that are even more difficult for the field of HCI alone to address. As we have outlined, new computer technologies and the transformations they are bringing about raise issues with much broader societal, moral, and ethical implications than HCI has had to deal with in the past. It is not clear that all of these concerns are within the scope of the field, but certainly HCI needs to be part of a wider interdisciplinary exchange. Technologies that store personal data, that take on new roles and responsibilities in our lives, that alter our behavior in public places, and that track our movements and

The latest billboards (such as those by Quividi) judge the gender and approximate age of people viewing them, with the potential of changing the nature of the advertisements they display. Technologies like these highlight the increasingly hybrid forms that interaction takes, as well as the scope of the "data" used to authenticate such interactions.

TOP: PHOTOGRAPH COURTESY OF INTERACTION RESEARCH STUDIO. BOTTOM: PHOTOGRAPH BY HIROKO MASUIKE activities are as much sociological as architectural and as much about politics as cognitive reasoning. Given the scope and complexity of these issues, HCI professionals need to engage in discourses that may at one time have seemed distant, if not entirely alien to them.

Redefining the H, C, and I. It is with these concerns in mind that the report suggests redefining the three elements of HCI—human, computer, and interaction.

The "H," representing the "user," clearly needs revision, especially given that people nowadays are as much consumers, creators, and producers as they are users of computers, and they often employ computers just for the fun of it. Conceptualizing the emotional aspects of experiencing technologies is already starting to happen. Words like magic, enchantment, pleasure, wonder, excitement, and surprise have begun to creep into the vocabulary when researchers and designers discuss the value of technology to people. But HCI specialists also need to ask what these terms really mean and how technologies may engender such experiences. The aesthetics of computational products has also gained importance in helping to define users' relationships to technology. Therefore new models would provide a better understanding of how the emotional aspects of computing relate to human values.

A new conception of the "C" in HCI is also needed so that we may better understand how the embedding of digital technologies in everyday objects, in the built structures around us, and in the natural landscape is transforming our surrounding environment into a physical-digital ecosystem. Thus we need to address not just the design of artifacts per se but also the spaces within which they reside. And the design has to deal with deeper and more systemic issues. As the computer becomes increasingly reliant on a larger world, and in particular as the connection to a network becomes an essential part of the computer's operation, the opportunity for improving the user experience simply through a better interface is rapidly disappearing. HCI needs concepts, frameworks, and methods that will enable it to consider people and computers as part of a messy world full of social, physical, technological, and physiological limitations and opportunities.

It follows that the "I" in HCI will also need to be understood at many different levels. As Greenfield[7] has so elegantly described, we will have to consider different sites of interaction—for example, interactions on and in the body, interactions between bodies, interactions between bodies and objects (properties such as graspable, pushable, and other human-centered descriptors may be important here), and interactions at the scale of kiosks, rooms, buildings, streets, and other public spaces. All these levels of interaction offer different physical and social "affordances"—readily perceivable action possibilities—that technologies can potentially change.

In redefining H, C, and I, and in extending what the field of HCI may achieve, we will need to develop a lingua franca that expresses not only new metaphors but also new principles. Such a common language will enable the diverse parties to better understand each other, to talk in detail about the emergent transformations, and to productively explore how to steer them in human directions.

In a world where people's movements and transactions can be tracked—where individuals trigger non-deliberate events just by being in a certain place, physical or virtual, at a certain time—the notion of interaction itself is being fundamentally altered. As the conception of technology use as a conscious act becomes difficult to sustain, other models of interaction and communication will have to be developed. At the other extreme, digital technologies will continue to be used in more deliberate and engaged ways as media for self-expression, community-building, identity-construction, self-presentation, and interpersonal relations. HCI professionals must understand the complexity of the new forms of social relations and interactions if they are to help develop technology that enables people's effective engagement.

The fact that we now live with technology and not just use it means that *HCI must also take into account the truly human element, conceptualizing "users" as embodied individuals who have desires and concerns and who function within a social, economic, and political ecology*. HCI must also be flexible, given that people's forms of engagement with technology and the nature of their interactions with it will continually be changing, often becoming more sophisticated, as they grow older. Understanding the new forms of interaction between humans and computers will involve asking questions about the qualitative—process, potential, and change—rather than quantifiable attributes and capabilities alone. C

References

1. Harper, R., Rodden, T., Rogers, Y., and Sellen, A. *Being Human: Human-Computer Interaction in the Year 2020.* Microsoft Research, Cambridge, U.K., 2008. Copies available on request at bhuman@microsoft.com.
2. Card, S., Moran, T., and Newell, A. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.
3. Cockton, G. A development framework for value-centered design. In *Proceedings of CHI '05, Extended Abstracts.* ACM Press, NY, 2005.
4. Dourish, P. *Where the Action Is: The Foundations of Embodied Interaction.* MIT Press, Cambridge, MA, 2001.
5. Friedman, B. and Kahn, P. H., Jr. Human values, ethics, and design. *Handbook on Human-Computer Interaction*, Jacko, J. and Sears, A., eds. Lawrence Erlbaum Associates, Mahwah, NJ, 2003, 1177–1201. (Revised second edition, 2008, 1241–1266.)
6. Friedman, B., Kahn, P. H., Jr., and Borning, A. Value-sensitive design and information systems. *Human-Computer Interaction in Management Information Systems: Foundations.* Zhang, P., and Galletta, D., Eds. M.E. Sharpe, Armonk, NY, 2006, 348–372.
7. Greenfield, A. *Everyware: The Dawning Age of Ubiquitous Computing.* Preachpit Press, NY, 2006.
8. Grudin, J. A moving target: The evolution of human-computer interaction. *Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*, Sears, A., and Jacko, J. Eds. Lawrence Erlbaum Associates, Mahwah, NJ, 2007, 1–24.
9. Harrison, S., Tatar, D., and Sengers, P. The three paradigms of HCI. In *Alt.chi. Proceedings of CHI '07.* ACM Press, NY, 2006.
10. McCarthy, J. and Wright, P. *Technology as Experience.* MIT Press, Cambridge, MA, 2004.
11. Norman, D.A. *Emotional Design: Why We Love (or Hate) Everyday Things.* Basic Books, NY, 2004.
12. Peters, J.D. *Speaking into the Air: A History of the Idea of Communication.* University of Chicago Press, Chicago/London, 1999.
13. Sproull, L. and Kiesler, S. *Connections: New Ways of Working in the Networked Organization.* MIT Press, Cambridge, MA, 1991.
14. Wendell, J., Wood, S., and Holtzblatt, K. *Rapid Contextual Design: A How-To Guide to Key Techniques for User-Centered Design.* Elsevier, 2004.

Abigail Sellen (asellen@microsoft.com) is Principal Researcher, Microsoft Research Cambridge, Cambridge, U.K.

Yvonne Rogers (yrogers@open.ac.uk) is a professor in the Department of Computing, The Open University, Milton Keynes, U.K.

Richard Harper (r.harper@microsoft.com) is Principal Researcher, Microsoft Research Cambridge, Cambridge, U.K.

Tom Rodden (tar@cs.nott.ac.uk) is a professor in the School of Computer Science, University of Nottingham, Nottingham, U.K.

DOI:10.1145/1467247.1467265

How avionics work led to a graphical language for reactive systems where the diagrams themselves define the system's behavior.

BY DAVID HAREL

Statecharts in the Making: A Personal Account

WRITING A HISTORICAL paper about something you yourself are heavily involved in is clearly difficult; the result is bound to be personal and idiosyncratic and might well sound presumptuous. I thus viewed an invitation to write about statecharts for the third History of Programming Languages conference in 2007 as an opportunity to put the language in a broader perspective.6 The present article is a greatly abridged version of the resulting conference paper. The implicit claim is that the emergence of the language brought to the forefront a number of ideas that today are central to software and systems engineering, including the general notions of visual formalisms, reactive systems, model-based development, model executability, and full code generation.

In 1979 I published a paper on a tree-like language based on the idea of alternation called "And/Or Programs,"[18] prompting Jonah Lavi of Israel Aircraft Industries (IAI) to contact me. We met in late 1982, at which time I'd been on the faculty of the Weizmann Institute of Science for two years. Lavi, who was a methodologist responsible for evaluating and recommending software engineering methods and tools at IAI, described some problems IAI avionics engineers were having. This was part of the massive effort then under way to build a fighter aircraft, the Lavi (no connection with Jonah's surname). Following that meeting, I began consulting at IAI on a one-day-a-week basis, and for several months Thursday became my IAI day.

The first few weeks were devoted to trying to understand the general issues from Lavi. Then it was time to be exposed to the details of the project and its specific difficulties, since I hadn't yet met the project's engineers. For several weeks, I spent my Thursdays with Lavi's assistant, Yitzhak Shai, and a select group of experts from the Lavi avionics team, notably Akiva Kaspi and Yigal Livne.

An avionics system is a wonderful example of what my colleague at Weizmann Amir Pnueli and I later identified as a reactive system.[17] The main behavior that dominates such a system is its reactivity, that is, its event-driven, control-driven, event-response nature. The behavior is often highly parallel and includes strict time constraints and possibly stochastic and continuous behavior. A typical reactive system is not predominantly data-intensive or algorithmic in nature. Behavior is the crucial problem in its development—the need to provide a clear yet precise description of what the system does or should do over time in response to both external and internal events.

The Lavi avionics team consisted of extremely talented people, including those involved in radar, flight control, electronic warfare, hardware, communication, and software. The radar people could provide the precise algorithm used to measure the distance to a target. The flight-control people could talk about synchronizing the controls in the cockpit with the flaps on the wings. The communications people could talk about formatting information traveling through the MuxBus communication line. Each had his own idiosyncratic ways of thinking about and explaining the system, as well as his own diagrams and emphases.

I would ask seemingly simple questions, such as: "What happens when this button is pressed?" In response, a weighty two-volume document would be brought out and volume A would be opened to page 389, clause 6.11.6.10, which says that if you press that button such then such a thing would occur. At which point (having by then learned some of the system's buzzwords) I would say: "Yes, but is that true even when an infrared missile is locked on a ground target?" To which someone might say, "Oh no, in volume B, page 895, clause 19.12.3.7, it says that in such a case this other thing happens." These Q&A sessions would continue, and when it would get to the fifth or sixth question the engineers were no longer sure of the answer and would have to call the customer (the Air Force people) for a response. By the time we got to the eighth or ninth question even the customer didn't have an answer.

Obviously, someone would eventually have to decide what happens when you press a certain button under a certain set of circumstances. However, this person might turn out to be a low-level programmer assigned to write some remote part of the code, inadvertently making decisions that influenced crucial behavior on a much higher level. Coming, as I did, from a clean-slate background in terms of avionics (a polite way of saying I knew nothing about the subject), this was shocking. It seemed extraordinary that such a talented and professional team knew in detail the algorithm used to measure the distance to a target but not many of the far more basic behavioral facts involving the system's response to a simple event.

To illustrate, consider the following three occurrences of a tiny piece of behavior buried in three totally different locations in a large specification of a chemical manufacturing plant:

"If the system sends a signal hot then send a message to the operator";

"If the system sends a signal hot with T >60° then send a message to the operator"; and

"When the temperature is maximum, the system should display a message on the screen, unless no operator is on the site except when T <60°."

Despite my formal education in mathematical logic, I've never been able to understand the third item. Sarcasm aside, the real problem is that all three were obviously written by three

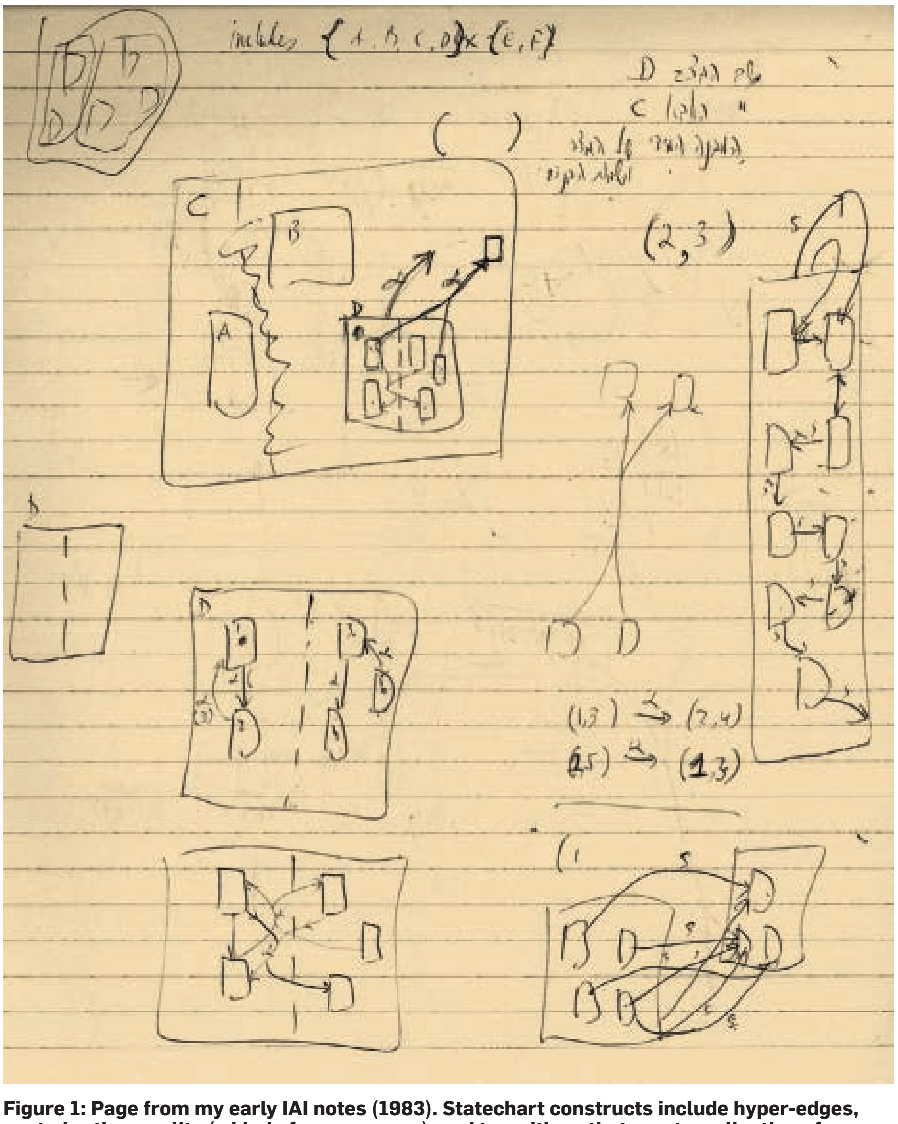


Figure 1: Page from my early IAI notes (1983). Statechart constructs include hyper-edges, nested orthogonality (a kind of concurrency), and transitions that reset a collection of states (chart on right). Note the use of Cartesian products of sets of states (set-theoretic formulation at the top) to capture the meaning of the orthogonality and the straightforward algebraic notation for transitions between state vectors (lower right).

different people for three different reasons. It is almost certain that the code for this critical aspect of the chemical plant would be problematic in the best case and catastrophic in the worst. The specification documents the Lavi avionics group had produced at IAI were no better. If anything, they were longer and more complex. Some subcontractors even refused to work from them, claiming they were incomprehensible, inconsistent, and incomplete.

This confusion prompted the question: How should an engineering team specify the behavior of such a complex reactive system in an intuitively clear yet mathematically rigorous fashion? This was what I aimed to try to answer.

How should an engineering team specify the behavior of such a complex reactive system in an intuitively clear yet mathematically rigorous fashion? This was what I aimed to try to answer.

Statecharts Emerging

My initial goal was not to invent a language but to try to recommend, from the toolset of the software and systems engineer, appropriate means for saying what the avionics engineers seemed to already have in mind. It turned out they really did understand the system and could answer many questions about behavior but often hadn't thought through things properly because the information wasn't well-organized in their documents (or heads). I had to spend a lot of the time getting them to talk; I kept asking questions, prodding them to state clearly how the aircraft behaves under certain sets of circumstances. We would then brainstorm, trying to make sense of the information that had piled up.

It was clear from the start that the basic idea of states/modes was fundamental to their way of thinking. (This insight was consistent with the work of David Parnas on the avionics of the A-7 jet fighter.[19]) The IAI avionics engineers would repeatedly say things like, "When the aircraft is in air-ground mode and you press this button, it goes into air-air mode, but only if the radar is not locked on a ground target." This is familiar to anyone in computer science; what we have here is really the likes of a finite-state automaton with its state transition mechanism. Nevertheless, having one big state machine describing what is going on would be fruitless, not only because of the exponentially growing number of states but also because simply listing all possible states and the transitions leading from one to the other is unstructured and nonintuitive; it provides no means for modularity, hiding of information, clustering, and separation of concerns and would not work for highly complex behavior. I was quickly convinced of the need for a structured and hierarchical extension of the conventional state machine formalism.

Following my initial attempt to use temporal-logic-like notation, I resorted to writing down the state-based behavior textually, in a kind of structured state-based dialect of "state protocols" made up on the fly (see the figures in [6]). The dialect was hierarchical; within a state there could be other states, and if you were in this state and the event occurred, you would enter the other state, and so on. As this went on, things would get increasingly complicated. The avionics engineers would bring up more of the system's behavior, and I would respond by extending the state-based structured description, often having to enrich the syntax in real time.

When the multitude of emerging behavioral details caused things to be even more complicated, I would doodle on the side of the page to explain (visually) what was meant. I recall the first time I used visual encapsulation to illustrate for the engineers the state hierarchy and an arrow emanating from the higher level to show a compound "leave-any-state" transition. I also recall the first time I used side-by-side adjacency with a dashed separator line to depict orthogonal (concurrent) state components (see Figure 1).

I drew these informal diagrams in order to explain what the nongraphical text-based state protocols meant. The text was still the real thing, however, and the diagrams were merely an aid. But after a while it dawned on me that everyone around the table seemed to understand these back-of-the-napkin diagrams much better, relating to them far more naturally. The pictures simply did a much better job of setting down on paper the system's behavior, as understood by the avionics engineers, and we found ourselves discussing the diagrams and arguing about the avionics over them, not over the state protocols. Still, the mathematician in me found this difficult to accept; I told myself that doodled diagrams could not really be better than a real mathematical-looking artifact. So it really took a leap of my own faith to be able to think: "Hmmm...couldn't the pictures be turned into the real thing, replacing, rather than supplementing, the textual structured-programming-like formalism?" So I gradually stopped using the text or used it only to capture supplementary information inside the states or along transitions, and the diagrams became the actual specification we were constructing.

This process of turning the diagrams into the specification language had to be done in a disciplined way, making sure the emerging pictures were not just pictures. You couldn't just throw in features because they looked good and the avionics team seemed to understand them. Unless the exact mathematical meaning of an intended feature was given—in any allowed context and under any allowed set of circumstances—it simply couldn't be considered. This was how the basics of the language emerged. I chose to use the term "statecharts" for the resulting creatures, which at the time was the only unused combination of "state" or "flow" with "chart" or "diagram" (see Figure 2).

The pilot stood there studying the blackboard for a couple of minutes, then said, "I think you have a mistake down here; this arrow should go over here and not over there." He was right.

Comments on the Language

Besides a host of secondary constructs, the two main ideas in statecharts—hierarchy and orthogonality—can be intermixed on all levels (see the figure). The language can be viewed as beginning with classical finite-state machines and their diagrams and extending them through a semantically meaningful hierarchical substating mechanism and through a notion of orthogonal simultaneity. Both extensions are reflected in the graphics—hierarchy by encapsulation of the blobs depicting states and orthogonality by partitioning a blob using dashed separator lines. Orthogonal components of the state space cooperate in several ways, including direct sensing of the state status in another component or through actions. The mechanism within a statechart thus has a broadcasting flavor whereby any part of the (same) statechart can sense what is happening in any other part.

As a result of the new constructs for hierarchy and orthogonality and their graphical renditions, transitions become far more elaborate and rich than in conventional state machines. They are still labeled with their triggering events and conditions but can now start or stop at any level of the hierarchy, cross levels, and in general be richer than standard source-target; they can be full hyperedges, since both sources and targets of transitions can be sets of states. At any given point in time a statechart is in a combination (vector) of states, the length of which is not fixed, since entering and exiting orthogonal components on various levels of the hierarchy changes the size of the state vector dynamically (see the nongraphical portions of the figure). Default states generalize start states, and the small arrows leading to them can be level-crossing and hyperedge in nature. In addition, statecharts also have special history connectors, conditions, output events, selection connectors, and more.

The fact that the technical part of the statechart story started out with And/Or Programs[18] is interesting and relevant. Encapsulated substates represent OR (actually XOR, exclusive or), and orthogonality is AND. Thus, a minimalist might view statecharts as a state-based language with an underlying structuring mechanism of classical alternation.

As for the graphic renditions, the two novel visual constructs in statecharts—blob encapsulation and partitioning—are both topological in nature and therefore worthy companions to edges in graphs. Indeed, when designing a graphical language, topology should be used whenever possible, since it is a more basic branch of mathematics than geometry. Being inside something is more fundamental and robust than being smaller or larger or than being a rectangle or a circle. Being connected to something is more basic than being green or yellow or being drawn with a thick line or a thin line. The human visual system notices and comprehends such things immediately.

Still, statecharts are not exclusively visual/diagrammatic. Transitions can be labeled not only with the events that cause the transitions but also with the conditions that guard against taking them and the actions (output events) that are to be carried out when they are taken. Moreover, statecharts borrow from both the Moore and the Mealy variants of state machines, allowing actions on transitions between states, as well as on entrances to or exits from states. Statecharts also allow "throughput" conditions attached to a state and are to hold through the entire time the system is in that state.

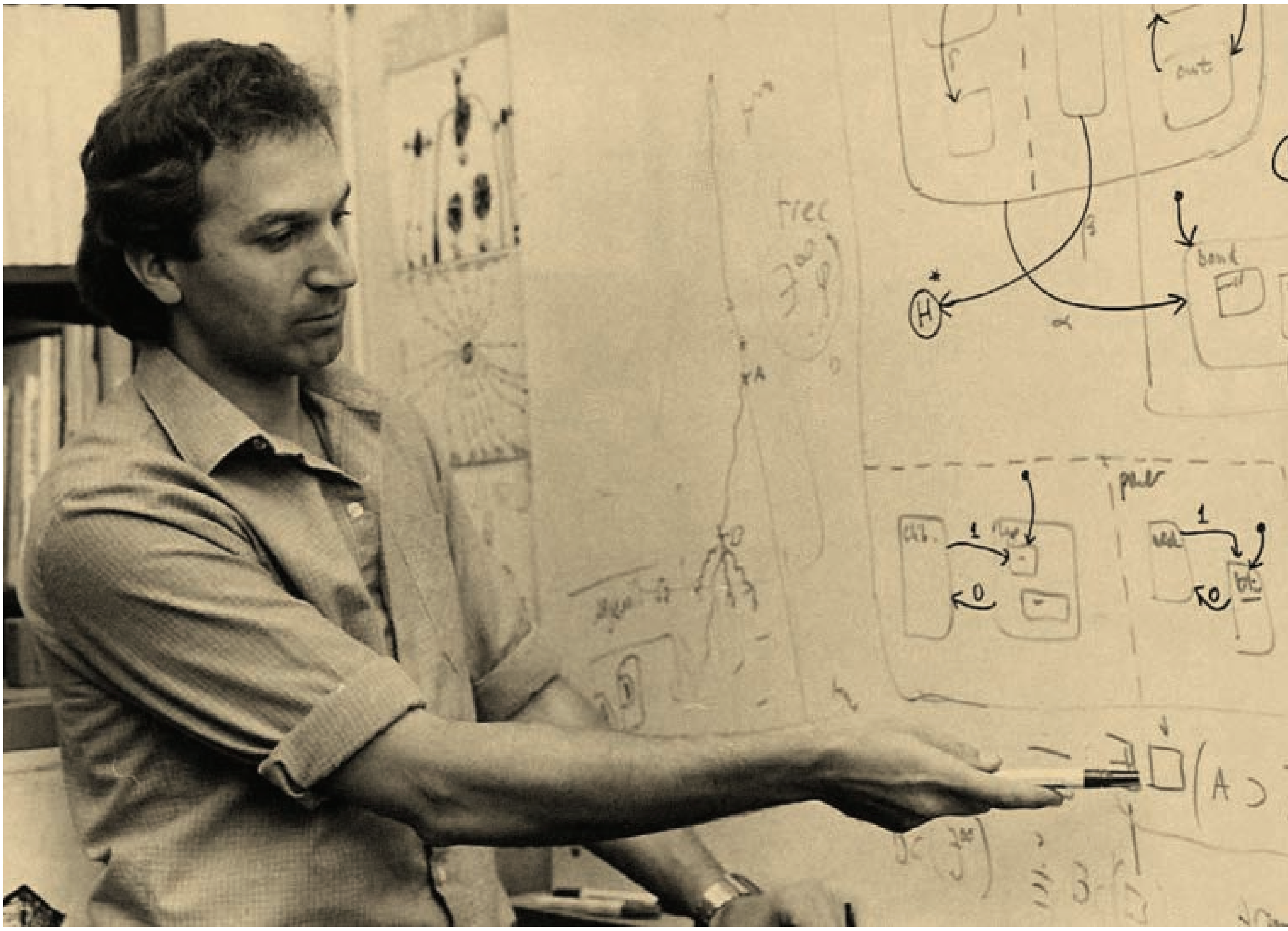


Figure 2: Explaining statecharts (1984); note the temporal logic bottom right.

Speaking in the strict mathematical sense of power of expression, hierarchy and orthogonality are but helpful abbreviations and can be eliminated; the hierarchy can be flattened, writing everything out explicitly on a low level, and orthogonality can be removed by taking the Cartesian product of the components (as in the top of the figure). Thus, these features do not add raw expressive power, and their value is reflected mainly in additional naturalness and convenience. However, they also (in general) provide great savings in size; for example, orthogonality can yield an exponential improvement in succinctness in both upper- and lower-bound senses.[3]

Incidentally, orthogonal state-components in statecharts do not necessarily represent concurrent or parallel components of the system being specified. They need not represent different parts of the system at all but can be introduced to help structure its state space and arrange the behavior in portions that are conceptually and intuitively separate, independent, and orthogonal. I emphasize the word "conceptually" because what counts is whatever is in the mind of the person doing the specification.

This motivation has many ramifications. I chose the broadcast communication mechanism of statecharts not because it is preferred for actual communication between a system's components. It is merely one way to coordinate the orthogonal components of the statechart, between its "chunks" of state-space, if you will; these will often not be the components—physical or software—of the system itself. Broadcasting is a way to sense in one part of the state space what is going on in another part and does not necessarily reflect actual communication between tangible aspects of the actual system. On certain levels of abstraction one often really wants to be able to sense properties of a part of the specification in another without worrying about implementation details. I definitely do not recommend having a single statechart for an entire system. Rather, as I discuss later, there will almost always be a decomposition of the system into functions, tasks, objects, and the like, each endowed with its own behavior (described by, for example, a statechart). In this way, the concurrency occurs on a higher level.

I return now to the two adjectives discussed earlier—"clear" and "precise"—behind the choice of the term "visual formalism."[14,16] Concerning clarity, the fact that a picture is worth a thousand words demands special caution. Not everything is beneficially depicted visually, but the basic topology-inspired graphics of statecharts seemed from the start to jibe well with the IAI avionics engineers; they quickly grasped the hierarchy and orthogonality, high- and low-level tran-

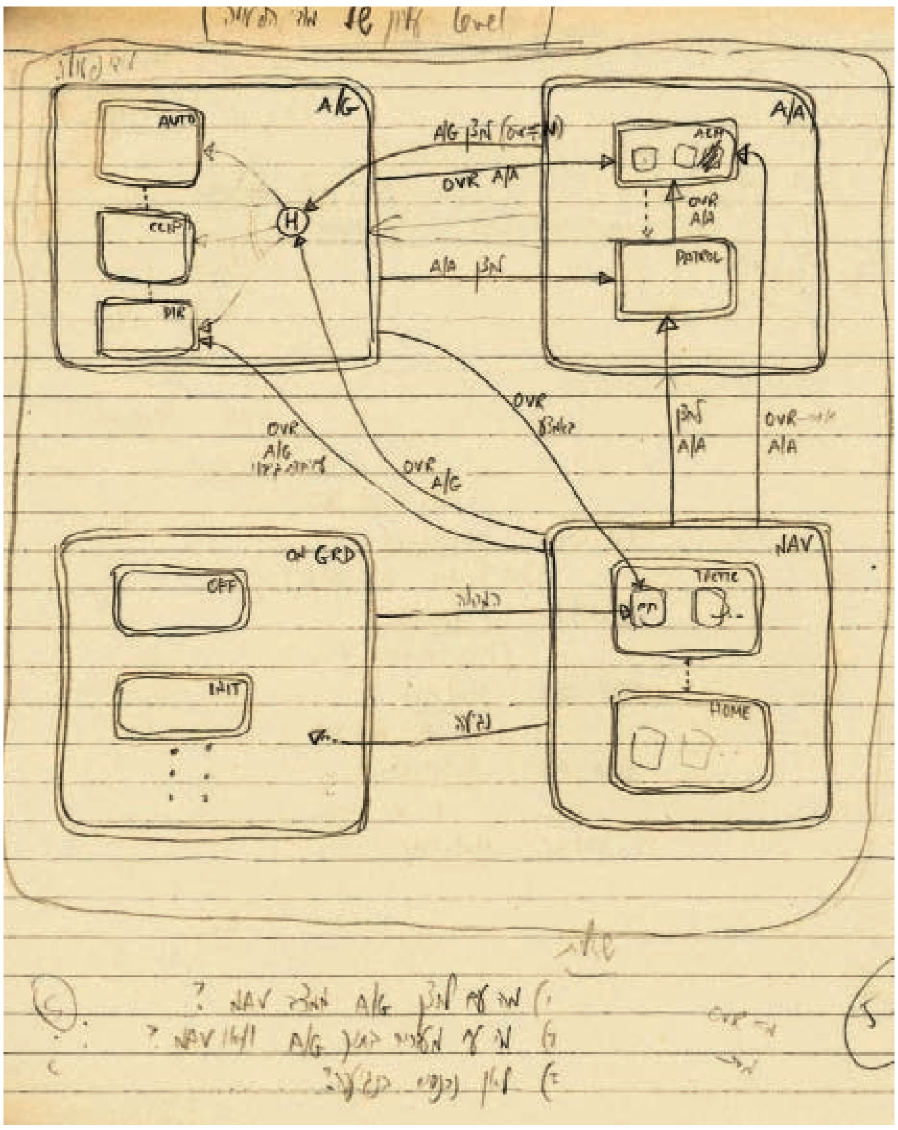


Figure 3: Page from the IAI notes (1983, events in Hebrew) showing a relatively "clean" draft of the top levels of behavior for the main flight modes of the Lavi avionics, including A/A (air-air), A/G (air-ground), NAV (automatic navigation), and ON GRD (on ground). Note early use of a history connector in the A/G mode.

sitions, default entries, and more.

Interestingly, the same quick comprehension applied to nonexperts outside the avionics group. I recall an anecdote from late 1983 in which in the midst of one session the blackboard showed a complicated statechart specifying the behavior of some intricate portion of the Lavi's avionics. A knock on the door brought in an Air Force pilot from the "customer" team who knew a lot about the aircraft being developed and its desired behavior but had never seen a state machine or a state diagram before, not to mention a statechart. I remember him staring at this intricate diagram (the statechart) on the blackboard, with its complicated mess of blobs inside other blobs, arrows splitting and merging, and asking, "What's that?" One of the engineers said, "That's the behavior of the so-and-so part of the system, and, by the way, these rounded rectangles are states, and the arrows are transitions between states." The pilot studied the blackboard for a couple of minutes, then said, "I think you have a mistake down here; this arrow should go over here and not over there." He was right.

For me, this little event indicated that we might be doing something right, that maybe what I was proposing was a good and useful way of specifying reactive behavior. If an outsider could come in, just like that, and grasp something that complicated without being exposed to the technical details of the language or the approach, then maybe we were on the right track (see Figure 3). Very encouraging.

So much for clarity. As for precision and formality, full executability was always central to the development of the language. I found it difficult to imagine the usefulness of a method that merely makes it possible to say things about behavior, give snippets of the dynamics or observations about what happens or what could happen, or provide some partially connected pieces of behavior. The whole idea was that if one builds a statechart-based specification everything must be rigorous enough to be run (executed) just like software written in a programming language. Executability was a basic, not-to-be-compromised, underlying concern during the process of designing the language. It might sound strange to a reader 26 years later, but in 1983 system-development tools did not execute models at all. Thus, turning doodles like those in the figure into a real language could be done only with great care.

Building a Tool

Once the basics of the language were established, it seemed natural to want a tool that could be used not only to prepare statecharts but also to execute them. So in April 1984, three colleagues (the brothers Ido and Hagi Lachover and Amir Pnueli) and I founded a company, Ad Cad, Ltd., later (1987) reorganizing it as I-Logix, Inc., with Ad Cad as its R&D branch. By 1986, we had built a tool for statecharts called Statemate.

In extensive discussions with the two most senior technical people associated with the company, Rivi Sherman and Michal Politi, along with Amir Pnueli, we were able to figure out during the Ad Cad period how to embed statecharts into a broader framework that was capable of capturing the structure and functionality of a large complex system. To this end, we proposed a diagrammatic language to structure a model that we called activity-charts, an enriched kind of hierarchical data-flow diagram whereby arrows represent the possible flow of information between the incident functions (activities). Each activity can be associated with a controlling statechart (or code) that would also be responsible for interfunction communication and cooperation.

Statemate also enabled one to specify the actual structure of the system itself, using module-charts that specify the components in the implementation of the system and their connections. In this way, the tool supported a three-way model-based development framework for systems consisting of structure, functionality, and behavior. The user could draw the statecharts and the model's other artifacts, link them together rigorously, check and analyze them, produce documents from them, and manage their configurations and versions. Most important, Statemate could fully execute them and generate from them, automatically, executable code in, say, Ada and C and later also in appropriate hardware description languages.

Even then, more than 20 years ago, Statemate could link the model to a GUI mockup of the system under development (or even to real hardware). Executability of the model could be done directly or by using the generated code and carried out in many ways with increasing sophistication. Verification wasn't in vogue in the 1980s, so analysis of the models was limited to various kinds of testing offered by Statemate in abundance. One could execute the model interactively (with the user playing the role of the system's environment), in batch mode (reading in external events from files) or in programmed mode. One could use breakpoints and random events to help set up and control a complex execution from which you could gather the results of interest. In principle, you could thus set up Statemate to "fly the aircraft" for you under programmed sets of circumstances, then come in the following day and find out what had happened. These capabilities, allowing us to "see" the model in operation, either via a GUI or following the statecharts as they were animated on the fly, were extremely useful to Statemate users. The tool was an eye-opener for software and systems engineers used to writing and debugging code in the usual way and was particularly beneficial for real-time and embedded systems.

Statemate is considered by some to be the first real-world tool to carry out true model executability and full code generation. The underlying ideas were the first serious proposal for model-driven system development. They might have been somewhat before their time but were deeply significant in bringing about the change in attitude that permeates modern-day software engineering, as exemplified by such efforts as the Unified Modeling Language. A decade after Statemate, we built the object-oriented Rhapsody tool at I-Logix (discussed later).

Woes of Publication

I wrote the first version of a paper describing statecharts in late 1983.[16] The process of trying to get it published was long and tedious but interesting in its own right. The details appear in the full version of the present article,[6] but I can say that the paper was rejected by several leading journals, including *Communications* and *IEEE Computer*. My files contain an interesting collection of referee comments and editor rejection letters, one of which asserted: "The basic problem [...] is that [...] the paper does not make a specific contribution in any area." It was only in July 1987 that the paper was finally published, in *Science of Computer Programming*.[16] The full version of the present article[6] also contains information (and anecdotes) about other publications on statecharts, including a paper I wrote with Pnueli defining reactive systems,[17] a *Communications* article on visual formalisms and higraphs,[14] an eight-author paper on Statemate,[13] the definitive paper on the Statemate semantics of statecharts,[13] and a Statemate book with Politi.[10]

Object-Oriented Statecharts

In the early 1990s, Eran Gery from I-Logix became interested in the work of James Rumbaugh and Grady Booch on the use of statecharts in an object-oriented framework. Gery did some gentle prodding to get me interested, with the ultimate result being a 1997 paper[11] in which we defined object-oriented statecharts and worked out the way we felt they should be linked up with objects and executed. In particular, we proposed two modes of communication between objects: direct synchronous invocation of methods and asynchronous queued events. The paper considered other issues, too, including creation and destruction of objects and multithreaded execution. The main structuring mechanism involved classes and objects, each of which could be associated with a statechart.

We we also outlined a new tool for supporting all this, and I-Logix promptly built it under the name Rhapsody.[11] One important difference between the function-oriented Statemate and the object-oriented Rhapsody is that the semantics of statecharts in Statemate is synchronous and in Rhapsody is (by and large) asynchronous. Another subtle but significant difference is reflected in the fact that Statemate was set up to execute statecharts directly in an interpreter mode separate from the code generator. In contrast, the model execution in Rhapsody is carried out by running the code generated from the model. A third difference is our decision to make the action language of Rhapsody a subset of the target programming language; for example, the events, conditions, and actions specified along state transitions are fragments of, say, C++ or Java. In any event, the statechart language may be considered a level higher than classical programming languages in that the code generated from it is in, say, C++, Java, or C; we thus might say that statecharts are high above C level.

Several software vendors have since developed tools based on statecharts or its many variants, including RoseRT (which grew out of ObjecTime), StateRover, and Stateflow (the statechart tool embedded in Matlab).

The implementation and tool-building issue can also be viewed in a broader perspective. In the early 1980s, no system-development tool based on graphical languages was able to execute models or generate full running code. Such CASE tools essentially consisted of graphic editors, document generators, and configuration managers and were thus like programming environments without a compiler. In contrast, I have always felt that a tool for developing complex systems must have the ability to not only describe behavior but also to analyze and execute it in full. This philosophy underlies the notion of a visual formalism, which must come endowed with sufficiently well-defined semantics so as to enable tools to be built around it that can carry out dynamic analysis, full model execution, and the automatic generation of running code.

On Semantics

It is worth dwelling further on the issue of semantics, which is a prerequisite for understanding the true meaning of any language, particularly executable ones. In a letter to me in 1984, Tony Hoare said that the statecharts language "badly needs a semantics." He was right. Being overly naïve at the time, I figured that writing a paper that explained the basics of the language's operation and then building a tool that executes statecharts and generates code from them would be enough. This approach took its cue from programming language research, where developers invent languages and then simply build compilers for them.

In retrospect, I didn't fully realize in those early years how different statecharts are from previous specification languages for real-time embedded systems. I knew, of course, that the language had to be executable, as well as easily understandable, even by people with no training in formal semantics. At the same time, as a tool-building team, we also had to demonstrate quickly to our sponsors, the first being IAI, that our efforts were economically viable. Due to the high level of abstraction of statecharts, we had to make decisions regarding rather deep semantic problems that apparently hadn't been adequately considered in the literature, at least not in the context of building a real-world tool intended for large, complex systems. Moreover, some of these issues were then being investigated independently by leading French researchers, including Gérard Berry, Nicholas Halbwachs, and Paul le Guernic, who coined the French phrase L'approche synchrone—the synchronous approach—for this kind of work.[1] Thus, when designing Statemate from 1984 to 1986, we did not do such a good job of deciding on the semantics.

We had to address a number of dilemmas regarding central semantic issues. One had to do with whether a step of the system should take zero time or more; another had to do with whether the effects of a step should be calculated and applied in a fixpoint-like manner in the same step or take effect only in the following step. The two issues are independent. The first concerns whether or not one adopts Berry's pure synchrony hypothesis,[1] whereby each step takes zero time. Clearly, these questions have many consequences in terms of how the language operates, whether events might interfere with chain reactions triggered by other events, how time itself is modeled, and how time interleaves with the system's discrete event dynamics.

At the time, we used the terms Semantics A and B to refer to the two main approaches we were considering. Both were synchronous in the sense of Benveniste et al.,[1] differing mainly in the second issue—regarding when the effects of a step take place. In Semantics A all events generated in the current step serve as inputs to the next step, whereas in Semantics B the system responds to all events generated internally in the current step until no further system response is possible. We called this chain of reactions a "super-step." The paper we published in 1987 was based on Semantics B,[15] but we later adopted semantics A for the Statemate tool itself.[10,13] Thus, Statemate statecharts constitute a synchronous language[1] and in that respect are similar to other, nonvisual languages in that family, including Esterel, Lustre (in commercial guise, known as Scade), and Signal.

We decided to implement Semantics A mainly because calculating the total effects of a step and carrying them out in the following step was easier to implement; we were also convinced that it was easier to understand for a typical systems engineer. Another consideration was related to the semantic level of compositionality; Semantics B strengthens the distinction between the system and its environment or between two parts of the system. If at some point in the development a system developer wants to consider part of the system to serve as an environment for the other part, the behaviors under Semantics B will be separated (as they should be), because chain reactions that go back and forth between the two halves are no longer all contained in a single super-step.

A number of other researchers had also begun looking into statechart semantics, often severely limiting the language (such as by completely dropping orthogonality) so the semantics are easier to define. Some of this work was motivated by the fact that our implemented semantics had not been published yet (we published in 1996[12]) and was not known outside the Statemate circle. This pre-object-oriented situation was summarized by Michael von der Beeck who tried to impose some order on the multitude of semantics of statecharts that were then being published. His resulting paper[22] claimed implicitly that statecharts are not well defined due to these many different semantics (it listed approximately 20). Interestingly, while [22] reported on the many variants of the language with their semantics, it did not report what should probably have been considered the language's "official" semantics, the one we defined and adopted in 1986 when building Statemate[13] but unfortunately also the only semantics not published at the time in the open public literature.

As to the semantic issues themselves, von der Beeck[22] discussed only the differences between variants of pre-object-oriented statecharts, but they are far less important than the differences between the non-object-oriented and the object-oriented versions of the language. The main semantic difference between Statemate and Rhapsody semantics is in synchronicity. In Statemate, the version of the statecharts language is based on functional decomposition and is a synchronous language, whereas the object-oriented-based Rhapsody version of statecharts is asynchronous. There are other substantial differences in modes of communication between objects; there are also issues arising from the presence of dynamic objects and their creation and destruction, inheritance, composition, and multithreading. The semantics of object-oriented statecharts was described in my 2004 paper with Hillel Kugler[8] (analogous to [12]) describing the differences between these two versions of the language.

Meanwhile, the Unified Modeling Language, or UML, which was standardized by the Object Management Group in 1997, featured many graphical languages, some of which are still not endowed with satisfactorily rigorous semantics. The heart of UML—its driving behavioral kernel—is the object-oriented version of statecharts. In the mid-1990s Eran Gery and I took part in helping the UML design team define the intended meaning of statecharts, resulting in UML statecharts being similar to those in [11] that we implemented in Rhapsody. For a manifesto about the subtle issues involved in defining the semantics of languages for reactive systems, see [7], with its whimsical subtitle "What's the Semantics of 'Semantics'?"

Biology, Hybrid Systems, Verification, Scenarios

Statecharts today are widely used in such application areas as aerospace, automotive, telecommunication, medical instrumentation, hardware design, and control systems. An interesting development also involves the language being used in such unconventional areas as modeling biological systems.[4,20,21]

Another important topic is hybrid systems. Statecharts can include probabilities, thus supporting probabilistic and stochastic behavior, but their underlying basis is discrete. It is very natural for a software or systems engineer to want to model systems with mixed discrete and continuous behavior, and it is not difficult to imagine using mathematics geared for continuous dynamics (such as differential equations) to model the activities within states in a statechart. An active community today is carrying out research on such systems.

Another topic involves exploiting the structuring of behavior in statecharts to aid verification of the modeled system. We all know how difficult program verification is, yet a number of techniques work well in many cases. While most common verification techniques do not exploit the hierarchical structure or modularity models often have, this structure can be used beneficially in verifying statecharts. Work has indeed been done on the verification of hierarchical state machines, though much more remains to be done.

Finally, I should mention some recent work my colleagues and I have carried out on a new approach to visual formalisms for complex systems. It involves a scenario-based specification method, rather than the state-based approach of statecharts. The idea is to concentrate on specifying the behavior between and among the objects (or tasks, functions, and components), not within them—inter-object rather than intra-object. The language we have proposed for this—Live Sequence Charts—was worked out jointly with Werner Damm.[2] The associated play-in and play-out programming techniques were developed later with my Ph.D. student Rami Marelly.[9] I published a paper last year describing a long-term vision on how this could be made much more general.[5]

Conclusion

If asked about the lessons to be learned from the statecharts story, I would definitely put tool support for executability and experience in real-world use at the top of the list. Too much computer science research on languages, methodologies, and semantics never finds its way into the real world, even in the long term, because these two issues do not get sufficient priority.

One of the most interesting aspects of this story is the fact that the work was not done in an academic tower, inventing something and trying to push it down the throats of real-world engineers. It was done by going into the lion's den, working with the people in industry. This is something I would not hesitate to recommend to young researchers; in order to affect the real world, one must go there and roll up one's sleeves. One secret is to try to get a handle on the thought processes of the engineers doing the real work and who will ultimately use these ideas and tools. In my case, they were the avionics engineers, and when I do biological modeling, they are biologists. If what you come up with does not jibe with how they think, they will not use it. It's that simple.

Looking back over the past 26 years, the main mistakes I made during the early years of statecharts concerned getting the message out to real-world software and systems engineers. This involved the confusing process of deciding on a clear semantics for the language and publicizing the chosen semantics promptly in the public literature, as well as not recognizing how important it was to quickly publish a book to acquaint engineers in industry with, and get them to use, a new language, method, and tool.

Nevertheless, despite the effort that went into developing the language (and later the tools to support it) I am convinced that almost anyone could have come up with statecharts, given the right background, exposure to the right kinds of problems, and right kinds of people.

Acknowledgment

Many people influenced this work, but my deepest gratitude goes to Jonah Lavi, Amir Pnueli, Eran Gery, Rivi Sherman, and Michal Politi.

References

1. Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Le Guernic, P., and de Simone, R. The synchronous languages 12 years later. *Proceedings of the IEEE 91* (2003), 64–83.
2. Damm, W. and Harel, D. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design 19*, 1 (2001), 45–80.
3. Drusinsky, D. and Harel, D. On the power of bounded concurrency I: Finite automata. *J. ACM 41* (1994), 517–539.
4. Efroni, S., Harel, D., and Cohen, I.R. Towards rigorous comprehension of biological complexity: Modeling, execution, and visualization of thymic T cell maturation. *Genome Research 13* (2003), 2485–2497.
5. Harel, D. Can programming be liberated, period? *IEEE Computer 41*, 1 (Jan. 2008), 28–37.
6. Harel, D. Statecharts in the making: A personal account. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages* (San Diego, CA, June 9-10). ACM Press, New York, 2007.
7. Harel, D. and Rumpe, B. Meaningful modeling: What's the semantics of 'semantics'? *IEEE Computer 37*, 10 (2004), 64–72.
8. Harel, D. and Kugler, H. The Rhapsody semantics of statecharts (or, on the executable core of the UML). In *Integrations of Software Specification Techniques for Applications in Engineering, LNCS, Vol. 3147*, H. Ehrig et al., Eds. Springer-Verlag, New York, 2004, 325–354.
9. Harel, D. and Marelly, R. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine.* Springer-Verlag, New York, 2003.
10. Harel, D. and Politi, M. *Modeling Reactive Systems with Statecharts: The Statemate Approach.* McGraw-Hill, New York, 1998.
11. Harel, D. and Gery, E. Executable object modeling with statecharts. *IEEE Computer 30*, 7 (1997), 31–42.
12. Harel, D. and Naamad, A. The statemate semantics of statecharts. *ACM Transactions on Software Engineering Methods 5*, 4 (Oct. 1996), 293–333.
13. Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtul-Trauring, A., and Trakhtenbrot, M. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering 16*, 4 (1990), 403–414.
14. Harel, D. On visual formalisms. *Commun. ACM 31*, 5 (May 1988), 514–530.
15. Harel, D., Pnueli, A., Schmidt, J., and Sherman, R. On the formal semantics of statecharts. In *Proceedings of the Second IEEE Symposium on Logic in Computer Science* (Ithaca, NY, 1987), 54–64.
16. Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, 3 (June 1987), 231–274.
17. Harel, D. and Pnueli, A. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, K.R. Apt, Ed. NATO ASI Series, Vol. F-13, Springer-Verlag, New York, 1985, 477–498.
18. Harel, D. And/Or programs: A new approach to structured programming. *ACM Transactions on Programming Languages and Systems 2*, 1 (Jan. 1980), 1–17.
19. Heninger, K.L., Kallander, J.W., Shore, J.E., and Parnas, D.L. *Software Requirements for the A-7E Aircraft, NRL Report 3876.* Washington, D.C., Nov. 1978.
20. Setty, Y., Cohen, I.R., Dor, Y., and Harel, D. Four-dimensional realistic modeling of pancreatic organogenesis. In *Proceedings of the National Academy of Science 105*, 51 (2008), 20374–20379.
21. Swerdlin, N., Cohen, I.R., and Harel, D. Toward an in-silico lymph node: A realistic approach to modeling dynamic behavior of lymphocytes. In *Proceedings of the IEEE, Special Issue on Computational System Biology 96*, 8 (2008), 1421–1443.
22. von der Beeck, M. A comparison of statecharts variants. In *Proceedings of Formal Techniques in Real Time and Fault Tolerant Systems, LNCS, Vol. 863.* Springer-Verlag, New York, 1994, 128–148.

David Harel (dharel@weizmann.ac.il) is the William Sussman Professorial Chair in the Department of Computer Science and Applied Mathematics at The Weizmann Institute of Science, Rehovot, Israel.

DOI:10.1145/1467247.1467267

Can a proof be checked without reading it?

BY MADHU SUDAN

Probabilistically Checkable Proofs

THE TASK OF verifying a mathematical proof is extremely onerous, and the problem is compounded when a reviewer really suspects a fatal error in the proof but still has to find an explicit one so as to convincingly reject a proof. Is there any way to simplify this task? Would not it be great if it were possible to scan the proof cursorily (such as, flip the pages randomly, reading a sentence here and a sentence there) and be confident that if the proof was buggy you would be able to find an error by such a superficial reading?

Alas, the current day formats of proofs do not allow such simple checks. It is possible to build a "proof" of any "assertion" (in particular ones that are not true) with just one single error, which is subtly hidden. Indeed, if you think back to the "proofs" of "1 = 2" that you may have seen in the past, they reveled in this flaw of current proof systems.

Fortunately this shortcoming of proofs is not an inherent flaw of logic. Over the past two decades, theoretical computer scientists have come up with novel formats for writing proofs of mathematical assertions. Associated with these formats are probabilistic algorithms that reviewers could (should?) use to verify the proofs. A reviewer using such a verification algorithm would be spared from reading the entire proof (and indeed will only read a "constant number of bits of the proof"—a notion we will elaborate on, and explain later). Any researcher who has a proof of a theorem can rewrite the proof in the prescribed format and offer it to the reviewer, with the assurance that the reviewer will be convinced of this new proof. On the other hand, if someone claims a faulty assertion to be a theorem, and offers any proof (whether in the new format or not), the reviewer will discover an error with overwhelming probability.

In what follows we will attempt to formalize some of the notions that we have been using in a casual sense above. The central notion that will emerge is that of a "probabilistically checkable proof (PCP)." Existence of such formats and verification algorithms often runs contrary to our intuition. Nevertheless they do exist, and we will discuss two different approaches that have been used thus far to construct such PCPs.

PCPs are arguably fascinating objects. They offer a certain robustness to the logical process of verification that may not have been suspected before. While the process of proving and verifying theorems may seem of limited interest (say, only to mathematicians), we stress that the notion of a "convincing" argument/evidence applies much more broadly in all walks of life. PCPs introduce the fascinating possibility that in all such cases, the time taken to assess the validity of the evidence in supporting some claims may be much smaller than the volume of the evidence. In addition to this philosophical interest in PCPs, there is a very different (and much more concrete) reason to study PCPs. It turns out that PCPs shed light in the area of combinatorial optimization. Unfortunately this light is "dark": The ability to construct PCPs mostly says that for many optimization problems where we knew that optimal solutions were (NP-) hard to find, even near-optimal solutions are hard to find. We discuss these connections soon after we discuss the definitions of PCPs.

Definitions

We start by formalizing what we mean by "theorems," "proofs," a "format" for proving them, and what it means to "change" such a format, that is, what changes are allowed, and what qualities should be preserved. Hopefully, in the process we will also manage to establish some links between the topic of this article and computer science.

To answer these questions we go to back to the essentials of mathematical logic. A system of logic attempts to classify "assertions" based on their "truth value," that is, separate true *theorems* from *false assertions*. In particular, theorems are those assertions that have "proofs." In such a system, every sentence, including theorems, assertions, and proofs, is syntactically just a finite string of letters from a finite alphabet. (Without loss of generality the alphabet may be binary, that is, $\{0, 1\}$.) The system of logic prescribes some axioms and some derivation rules. For an assertion to be true, it must be derivable from the axioms by applying a sequence of derivation rules. A *proof* of an assertion A may thus be a sequence of more and more complex assertions ending in the assertion A, with each intermediate assertion being accompanied with an explanation of how the assertion is derived from previous assertions, or from the axioms. The exact set of derivation rules used and the complexity of a single step of reasoning may vary from one logical system to another, but the intent is that eventually all logical systems (based on the same axioms) should preserve two essentials aspects: The set of theorems provable in any given system of logic should be the same as in any other. Furthermore, proofs should be "easy" to verify in each.

This final attempt to abstract the nature of a logical system leaves us with the question: What is "easy?" It is this aspect that led to the development of Turing and Church's work on the Turing machine. They ascribed "easiness" to being a mechanical process, as formalized by the actions of some Turing machine. Modern computational complexity is little more careful with this concept. The task of verification of a proof should not only be "mechanical," but also "efficient," that is, should be polynomial time computable. This leads to the following abstract notion of a system of logic: A system of logic is given by a polynomial time verification algorithm (or simply *verifier*) $V(\cdot, \cdot)$, that takes two inputs, an assertion A and some evidence E and produces a Boolean verdict "`accept/reject`." If $V(A, E) =$ `accept` then and E is a proof of A. If A is an assertion such that there exists some E such that $V(A, E) =$ `accept`, then A is a theorem. In contrast to the notion that a proof is easy to verify, our current state of knowledge suggests that proofs may be hard to find, and this is the essence of the theory of NP-completeness.[11, 25, 27] Indeed the question "is NP = P?" is equivalent to the question "can every theorem be proved efficiently, in time polynomial in the length of its shortest proof?"

In what follows we will fix some system of logic, that is, some verifier V_0 and consider other verifiers that are *equivalent* to this verifier. In such cases, when the set of theorems does not change, but the "proofs" may, we call the new system a "proof system." So, a proof system V would be equivalent to V_0 if the following conditions hold:

Completeness: If A is a theorem (in V_0), then A is a theorem in V. Furthermore, there is a proof of A in V that is at most a polynomial factor longer than its proof in V_0.
Soundness: If A is not a theorem (in V_0), then A is not a theorem in V.

By allowing different verifiers, or proof systems, for the same system of logic, one encounters many different ways in which theorems can be proved. As an example, we show how the NP-completeness of the famous problem 3SAT allows one to produce formats for proofs that "localize" errors in erroneous proofs. Recall that an instance of 3SAT is a logical formula $\phi = C_1 \wedge \cdots C_m$ where C_j is the disjunction of three literals (variables or their complement). The NP-completeness of 3SAT implies the following: Given any assertion A and integer N, there is a 3CNF formula ϕ (of length bounded by a polynomial in N) such that ϕ is satisfiable if and only if A has a proof of length at most N (in V_0). Thus the deep logical question about the truth of A seems to reduce to a merely combinatorial question about the satisfiability of ϕ. The natural evidence for the satisfiability of ϕ would be an assignment and this is what we refer to as a "format" for proofs. The advantage of this format is that in order to reject an "erroneous proof," that is, an assignment x that fails to satisfy ϕ, one only needs to point to one clause of ϕ that is not satisfied and thus only point to the three bits of the proof of x that this clause depends on to reveal the error. Thus errors are easily localized in this format.

Can one go further and even "find" this error efficiently? This is where PCPs come in. In what follows, we will attempt to describe verifiers that can verify proofs of satisfiability of 3CNF formulae (noticing that by the discussion above, this is as general as verifying proofs of any mathematical statement in any formal system).

Probabilistically checkable proofs. We start by formalizing the notion of the number of bits of a proof that are "read" by the verifier. In order to do so, we allow the verifier to have random access (oracle access) to a proof. So while the proof may be a binary string $\pi = \langle \pi[1] \ldots \pi[\ell] \rangle \in \{0,1\}^\ell$, the verifier gets to "read" the ith bit of π by querying an "oracle" for the ith bit of π and get $\pi[i]$ in response, and this counts as one *query*.

PCPs are motivated by the question: "how many bits of queries are really essential to gain some confidence into the correctness of a theorem?" It is easy to argue that if the verifier hopes to get away by querying only a constant number of bits of the proof, then it cannot hope to be deterministic (else a constant time brute force search would find a proof that the verifier would accept). So we will allow the verifier to be probabilistic, and also it to make mistakes (with low probability). This leads us to the notion of a PCP verifier.

Definition 2.1. *A PCP verifier of query complexity $q(n)$, and gap $\varepsilon(n)$ is a probabilistic algorithm V that, given as input an assertion $A \in \{0, 1\}^n$, picks a random string $R \in \{0, 1\}^*$, makes oracle queries to a proof oracle $\pi: \{1,\ldots,\ell\} \to \{0, 1\}$ and produces an accept/reject verdict.*

Running time: *V always runs in time polynomial in n.*
Query complexity: *V makes $q(n)$ queries into the proof π.*
Proof size: *ℓ grows polynomial in n.*
Completeness: *If A is a true assertion, then there exists a proof π that the verifier accepts on every random string R.*
Soundness, with gap $\varepsilon(n)$: *If A is not true, then for every proof π, the probability, over the choice of the randomness R, that the verifier accepts at most $1 - \varepsilon(n)$.*

The above definition associates two parameters to a PCP verifier, query complexity, and gap. The query complexity is the number of bits of the proof that the verifier "reads." The gap is related to the "error" probability of the verifier, that is, the probability of accepting false assertions. The larger the gap, the smaller the error. Since the definition above introduces several notions and parameters at once, let us use a couple of simple examples to see what is really going on.

The classical proof of satisfiability takes a formula of length n, on up to n variables and gives a satisfying assignment. The classical verifier, who just reads the entire assignment and verifies that every clause is satisfied, is also a PCP verifier. Its query complexity $q(n)$ is thus equal to n. Since this verifier makes no error, its gap is given by $\varepsilon(n) = 1$.

Now consider a probabilistic version of this verifier who chooses to verify just one randomly chosen clause of the given 3CNF formula. In this case the verifier only needs to query three bits of the proof and so we have $q(n) = 3$. How about the gap? Well, if a formula is not satisfiable, then at least one of the up to n clauses in the formula will be left unsatisfied by every assignment. Thus once we fix a proof π, the probability that the verifier rejects is least $1/n$, the probability with which the verifier happens to choose a clause that is not satisfied by the assignment π. This corresponds to a gap of $\varepsilon(n) = 1/n$. (Unfortunately, there do exist (many) unsatisfiable 3CNF formulae which have assignments that may satisfy all but one clause of the formula. So the gap of the above verifier is really $\Theta(1/n)$.)

Thus PCP verifiers are just extensions of "classical verifiers" of proofs. Every classical verifier is a PCP verifier with high query complexity and no error (that is, high gap), and can be converted into one with low (constant!) query complexity with high error (tiny gap). Indeed a smooth trade-off between the parameters can also be achieved easily. To reduce the error (increase the gap) of a PCP verifier with query complexity q and gap ε, we could just run this verifier several, say k, times on a given proof, and reject if it ever finds an error. The new query complexity is now kq and if the theorem is not true then the probability of detecting an error is now $(1 - \varepsilon)^k$. The new error is approximately $1 - k\varepsilon$ if $k \ll 1/\varepsilon$ and thus the gap goes up by a factor of roughly k.

The fundamental question in PCP research was whether this trade-off was essential, or was it possible to get high gap without increasing the number of queries so much. The PCP theorem states that such proof systems can be constructed!

Theorem 2.2 (PCP Theorem[3, 4, 12]). *3SAT has a PCP verifier of* constant *query complexity and* constant *positive gap*.

There are two distinct proofs of this theorem in the literature and both are quite nontrivial. In Section 4 we will attempt to give some idea of the two proofs. But before that we will give a brief history of the evolution of the notion of a PCP, and one of the principal applications of PCPs in computer science. The exact constants (in the query complexity and gap) are by now well studied and we will comment on them later in the concluding section.

History of definitions. The notion of a PCP verifier appears quite natural given the objective of "quick and dirty" verification of long proofs. However, historically, the notion did not evolve from such considerations. Rather the definition fell out as a byproduct of investigations in cryptography and computational complexity, where the notion of a PCP verifier was one of many different elegant notions of probabilistic verification procedures among interacting entities. Here we attempt to use the historic thread to highlight some of these notions (see Goldreich[18] for a much more detailed look into these notions). We remark that in addition to leading to the definition of PCPs, the surrounding theory also influences the constructions of PCP verifiers—indeed one may say that one may have never realized that the PCP theorem may be true, had it not been for some of the prior explorations!

Interactive Proofs: The first notion of a probabilistic proof system to emerge in the literature was that of an *interactive proof*. This emerged in the works of Goldwasser et al.[20] and Babai and Moran.[7] An interactive proof consists of an interaction between two entities (or agents), a "prover" and a "verifier." The prover wishes to convince the verifier that some theorem A is true. The verifier is again probabilistic and runs in polynomial time, and should be convinced if the assertion is true and should reject any interaction with high probability if the assertion is not true.

The goal here was not to improve on the efficiency with which, say, proofs of 3-satisfiability could be checked. Instead the goal was to enhance the class of assertions that could be verified in polynomial time. A nonmathematical, day-to-day, example of an interactive proof would be that of distinguishing between two drinks. Imagine convincing your spouse or friend that buying an expensive bottle of wine, brand X, is really worth it. They may counter with a cheap bottle, brand Y, that they claim tastes exactly the same. You insist that they taste quite different, but it is hard to prove your point with any written proof. But this is something we could attempt to prove interactively, by a *blind taste* test. You can ask your spouse/friend to challenge you with a random glass of wine and if by tasting you can tell which brand it is, you manage to convince your partner that you may have a point—the two drinks do taste different. By repeating this test many times, you partner's conviction increases. Interactive proofs attempt to such capture phenomena and more. Indeed, this very example is converted to a very mathematical one by Goldreich et al.,[19] who use a mathematical version here to give proofs that two graphs are distinguishable, that is, they are not isomorphic. (This is a problem for which we do not know how to give a polynomially long proof.)

The initial interest in interactive proofs came from two very different motivations. Goldwasser et al. were interested in the "knowledge" revealed

One of the somewhat strange aspects of PCP research has been that even though the existence of PCPs seems to be a "positive" statement, its use is mostly negative. We suggest that positive uses might emerge as more of our life turns digital, and we start worrying not only about the integrity of the data, but some of the properties they satisfy.

in multiparty interactions, from the point of view of maintaining secrets. To understand this concept, they first needed to define interactive protocols and interactive proofs, and then a formal measure of the knowledge complexity of this interaction. They noted that while interaction may reveal many bits of "information" (in the sense of Shannon[34]) to the interacting players, it may reveal little knowledge. For example, the interactive proof above that brand X is distinguishable from brand Y reveals no more "knowledge" than the bare fact that they are distinguishable. It does not, for example, tell you what features are present in one brand and not in the other.

Babai and Moran's motivation was more oriented towards computational complexity of some number-theoretic and group-theoretic problems. They were able to present interactive proofs with just one rounds of interaction between verifier and the prover for a number of problems not known to be in NP (i.e, not reducible to satisfiability). The implication, proved formally in later works, was that such problems may not be very hard computationally.

The theory of interactive proofs saw many interesting discoveries through the 1980s, and then culminated in a surprising result in 1990 when Shamir,[33] based on the work of Lund et al.,[28] showed the set of assertions that could be proved interactively were exactly those that could be verified by a polynomial space bounded verifier.

Multi-prover and Oracle-Interactive Proofs: Part of the developments in the 1980s led to variations on the theme of interactive proofs. One such variation that became significant to the development of PCPs was the notion of a "Multi-prover Interactive Proof" (MIP) discovered by Ben-Or et al.[8] Ben-Or et al. were trying to replace some cryptographic assumptions (along the lines of statements such as "RSA is secure") in existing interactive proofs with non-cryptographic ones. This led them to propose the study of the setting where the proof comes from a pair of provers who, for the purpose of the verification task, are willing to be separated and quizzed by the verifier. The hope is that the verifier can quiz them on related facts to detect inconsistency on their part. This limits the prover's ability to cheat and Ben-Or et al. leveraged this to create protocols where they reveal little knowledge about the proof when trying to prove their assertion to the verifier. (Some of the information being leaked in single-prover protocols was occurring because the prover needed to prove its honesty, and this information leakage could now stop.)

Fortnow et al.[16] tried to study the power of multiprover interactions when the number of provers increased from two to three and so on. They noticed that more than two provers does not enhance the complexity of the assertions that could be proved to a polynomial time verifier. Key to this discovery was the notion of an "Oracle-Interactive Proof." This is yet another variation in the theme of interactive proofs where the prover is "semi-honest." Specifically, the prover behaves as an oracle—it prepares a table of answers to every possible question that may be asked by the verifier and then honestly answers any sequence of questions from the verifier according to the prepared set of answers. (In particular, even if the history of questions suggests that a different answer to the latest question may increase the probability that the verifier accepts, the oracle does not change its mind at this stage.) Fortnow et al. noted that Oracle-interactive proofs simulate any (polynomial) number of provers, and are in turn simulated by 2-prover proof systems with just one round of interaction (that is, the verifier asks each of the two provers one question each, without waiting for any responses from the provers, and then the provers respond).

Subsequent to Shamir's result on the power of IP (single prover interactive proofs), Babai et al.[6] gave an analogous characterization of the power of MIP. They showed that the set of assertions that can be proved in polynomial time to a probabilistic verifier talking to two provers is the same as the set of assertions that could be verified in exponential time by a deterministic (classical) verifier. Thus the interaction with multiple provers reduced verification time from exponential to a polynomial!

Holographic Proofs, PCPs: In subsequent work, Babai et al.[5] noticed that the notion of an oracle-interactive proof was not very different from the classical notion of a proof. If one considers the table implied by the oracle prover and writes it down explicitly, one would get a very long string (potentially exponentially long in the running time of the verifier), which in effect was attempting to prove the assertion. In the result of Babai et al.,[6] this oracle-based proof is not much longer than the classical proof (both have length exponential in the length of the assertion), but the oracle proof was much easier to check (could be checked in polynomial time). This led Babai et al. to name such proofs *holographic* (small pieces of the proof reveal its correctness/flaws). Babai et al. focussed on the computation time of the verifier and showed (in some careful model of verification) that every proof could be converted into a holographic one of slightly superlinear size, where the holographic one could be verified by the verifier in time that was some polynomial in the logarithm of the length of the proof.

Around the same time, with a very different motivation that we will discuss in the next section, Feige et al.[15] implicitly proposed the concept of a *PCP* with the emphasis now being on the query complexity of the verifier (as opposed to the computation time in holographic proofs). The notion of PCP was finally explicitly defined by Arora and Safra.[4]

We stress that the theory of PCPs inherits much more than just the definition of PCPs from the theory of interactive proofs. The results, techniques, and even just the way of thinking, developed in the context of interactive proofs played a major role in the development of the PCP theorem. In particular, the notion of 2-player 1-round interactive proof and their equivalence to oracle-interactive proofs and hence PCPs plays a significant role in this theorem and we will use this notion to explain the proofs from a high level.

Implications to Combinatorial Optimization

The notion of theorems and proofs has shed immense light on the complexity of combinatorial optimization. Consider a prototypical problem, namely graph coloring, that is, the task of coloring the vertices of an undirected graph with the minimum number of possible colors so that the endpoints of every edge have distinct colors. The seminal works of Cook, Levin, and Karp[11, 25, 27] show that this task is as hard as finding a proof of some given theorem. In other words, given an assertion A and estimate N on the length a proof, one can construct a graph G on $O(N^2)$ vertices with a bound K, such that G has a coloring with K or fewer colors if and only if A has a proof of length at most N. Furthermore, given a K-coloring of G, one can reconstruct a proof of A in time polynomial in N. Thus unless we believe that proofs of theorems can be found in time polynomial in the length of the shortest proof (something that most mathematicians would find very surprising), we should also believe that graph coloring cannot be solved in polynomial time.

Of course, graph coloring is just one example of a combinatorial optimization problem that was shown by the theory of NP-completeness to be as hard as the task of theorem-proving. Finding large independent sets in graphs, finding short tours for travelling salesmen, packing objects into a knapsack are all examples of problems for which the same evidence of hardness applies (see Garey[17] for many more examples). The NP-completeness theory unified all these problems into the same one, equivalent to theorem-proving.

Unfortunately, a somewhat more careful look into the different problems revealed many differences among them. This difference became apparent when one looked at their "approximability." Specifically, we say that an algorithm A solves a (cost) minimization problem Π to within some approximation factor $\alpha(n)$ if on every input x of length n, $A(x)$ outputs a solution whose cost is no more than $\alpha(n)$ factor larger than the minimum cost solution. For (profit) maximization problems, approximability is defined similarly: An $\alpha(n)$ approximation algorithm should produce a solution of cost at least the optimum divided by $\alpha(n)$. Thus $\alpha(n) \geq 1$ for every algorithm A and problem Π.

The NP-completeness theory says that for the optimization problems listed above find a 1-approximate solution (that is, the optimum one) is as hard as theorem-proving. However, for some NP-complete minimization problems, it may be possible to find a solution of cost, say, at most twice the optimum in polynomial time for every input. Indeed this happens for the travelling salesman problem on a metric space (a space where distances satisfy triangle inequality). If one finds a minimum cost spanning tree of the graph and performs a depth-first-traversal of this tree, one gets a "tour" that visits every node of the graph at least once and has a cost of at most twice the cost of the optimum travelling salesperson tour. (This tour may visit some vertices more than once, but such extra visits can be short-circuited. The short-circuiting only produces a smaller length tour, thanks to the triangle inequality.) Thus the travelling salesman problem with triangle inequalities ($\triangle$-TSP) admits a polynomial time 2-approximation algorithm. Does this imply that every optimization problem admits a 2-approximation algorithm? Turns out that not even a $\sqrt{n}$-approximation algorithm is known for graph coloring. On the other hand, the knapsack problem has a $(1 + \varepsilon)$-approximation algorithm for every positive ε, while the same is not known $\triangle$-TSP. Thus while the theory of NP-completeness managed to unify the study of optimization problems, the theory of "approximability" managed to fragment the picture. Till 1990 however it was not generally known if the inability to find better approximation algorithms was for some inherent reason, or was it merely our lack of innovation. This is where the PCP theory came in to the rescue.

In their seminal work, Feige et al.[15] came up with a startling result. They showed that the existence of a PCP verifier as in the PCP theorem (note that their work preceded the PCP theorem in the form stated here, though weaker variants were known) implied that if the independent set size in a graph could be approximated to within any constant factor then NP would equal P! Given a PCP verifier V and an assertion A, they constructed, in polynomial time, a graph $G = G_{V,A}$ with the property that every independent set in G corresponded to a potential "proof" of the truth of A, and the size of the independent set is proportional to the probability with which the verifier would accept that "proof." Thus if A were true, then there would be a large independent set in the graph of size, say K. On the other hand, if A were false, every independent set would be of size at most $(1 - \varepsilon)K$. Thus the gap in the acceptance probability of the verifier turned into a gap in the size of the independent set. A $1/(1 - \varepsilon/2)$-approximation algorithm would either return an independent set of size greater than $(1 - \varepsilon/2)K$, in which case A must be true, or an independent set of size less than $(1 - \varepsilon)K$ in which case we may conclude that A is false. Thus a $1/(1 - \varepsilon/2)$-approximation algorithm for independent sets suffices to get an algorithm to decide truth of assertions, which is an NP-complete task.

The natural next question is whether the connection between independent set approximation and PCPs is an isolated one—after all different problems do behave very differently with respect to their approximability, so there is no reason to believe that PCPs would also yield inapproximability results for other optimization problems. Fortunately, it turns out that PCPs do yield inapproximability results for many other optimization problems. The result of Feige et al. was followed shortly thereafter by that of Arora et al.[3] who showed that for a broad collection of problems, there were nontrivial limits to the constant factor to which they were approximable, unless NP = P. (In other words, for each problem under consideration they gave a constant $\alpha > 1$ such that the existence of an α-factor approximation algorithm would imply NP = P.) This collection was the so-called MAX SNP-hard problems. The class MAX SNP had been discovered earlier by Papadimitriou and Yannakakis[30] and their work and subsequent works had shown that a varied collection of problems including the MAX CUT problem in graphs, Vertex Cover problem in graphs, Max 3SAT (an optimization version of 3SAT where the goal is to satisfy as many clauses as possible), $\triangle$-TSP, Steiner trees in metric spaces, the shortest superstring problem were all MAX SNP-hard. Subsequently more problems were added to this set by Lund and Yannakakis[29] and Arora et al.[1] The combined effect of these results was akin to that of Karp's work[25] in NP-completeness. They suggested that the theory of PCPs was as central to the study of approximability of optimization problems, as NP-completeness was to the exact solvability of optimization problems. Over the years, there have been many successful results deriving inapproximability results from PCP machinery for a wide host of problems (see surveys by Arora and Khot[2, 26] for further details). Indeed the PCP machinery ended up yielding not only a first cut at the approximability of many problems, but even very tight analyses in many cases. Some notable results here include the following:

- Håstad[22] showed that Max 3SAT does not have an α-approximation algorithm for $\alpha < 8/7$. This is tight by a result of Karloff and Zwick[24] that gives an 8/7 approximation algorithm.
- Feige[14] gave a tight inapproximability result for the Set Cover problem.
- Håstad[21] shows that the clique size in n-vertex graphs cannot be approximated to within a factor of $n^{1-\varepsilon}$ for any positive ε.

Again, we refer the reader to some of the surveys for more inapproximability results[2, 26] for further details.

Construction of PCPs: A Bird's Eye View

We now give a very high level view of the two contrasting approaches towards the proof of the PCP theorem. We stress that this is not meant to give insight, but rather a sense of how the proofs are structured. To understand the two approaches, we find it useful to work with the notion of 2-prover one round proof systems. While the notion is the same as the one defined informally in Section 2, here we define the verifier more formally, and introduce the parameter corresponding to query complexity in this setting.

Definition 4.1 (2IP verifier, Answer size, Gap). *A* 2IP verifier *of answer size $a(n)$, and gap $\varepsilon(n)$ is a probabilistic algorithm V who, on input an assertion $A \in \{0, 1\}^*$, picks a random string $R \in \{0,1\}^*$, makes one query each to two provers $P_1, P_2 : \{1, \ldots, \ell\} \to \{0, 1\}^*$, and produces an* `accept/reject` *verdict, denoted $V^{P_1, P_2}(A; R)$, with the following restrictions, when $A \in \{0, 1\}^n$:*

Running time: *V always runs in time polynomial in n.*
Answer size: *The prover's answers are each $a(n)$ bits long.*
Prover length: *The questions to the provers are in the range $\{1,\ldots,\ell(n)\}$ where $\ell(n)$ is a polynomial in n.*
Completeness: *If A is a true assertion, there exist provers P_1, P_2 such that $V^{P_1, P_2}(A; R)$ always accepts.*
Soundness, with gap $\varepsilon(n)$: *If A is not true, then for every pair of provers P_1, P_2 the probability, over the choice of the random-*

ness R, that $V^{P_1,P_2}(A;R)$ *outputs* `accept` *is at most* $1 - \varepsilon(n)$.

The definition is (intentionally) very close to that of a PCP verifier, so let us notice the differences. Rather than one proof oracle, we now have two provers. But each is asked only one question, so effectively they are oracles! In PCPs, the response to a query is one bit long, but now the responses are $a(n)$ bits long. On the other hand, in PCPs, the verifier is allowed to make $q(n)$ queries to the proof oracle, while here the verifier is only allowed one query each. Nevertheless, PCP verifiers and MIP verifiers are closely related. In particular, the following proposition is really simple from the definitions.

PROPOSITION 4.2. *If 3SAT has a 2IP verifier of answer size a(n) and gap* $\varepsilon(n)$, *then 3SAT has a PCP verifier with query complexity* $2 \cdot a(n)$ *and gap* $\varepsilon(n)$.

The PCP verifier simply simulates the 2IP verifier, with each query to the provers being simulated by $a(n)$ queries to the proof oracle.

Thus to prove the PCP theorem, it suffices to give a 2IP verifier with constant gap and constant answer size. We start with the approach of Arora and Safra.[4]

Reducing answer size. The initial proofs of the PCP theorem approached their goal by holding the gap to be a constant, while allowing the 2IP verifier to have (somewhat) long answer sizes. Key to this approach were some alphabet reduction lemmas initiated in the work of Arora and Safra.[4] Here we state two from Arora et al.,[3] that suffice for our purposes.

LEMMA 4.3 (ARORA ET AL.[3]). *There exists a constant* $\delta > 0$ *such that if 3SAT has a 2IP verifier with answer size a(n) and gap* ε, *then 3SAT also has a 2IP verifier with answer size* $(\log a(n))^2$ *and gap* $\varepsilon \cdot \delta$.

LEMMA 4.4 (ARORA ET AL.[3]). *There exist constants* $c < \infty$ *and* $\tau > 0$ *such that if 3SAT has a 2IP verifier with answer size* $a(n) = o(\log n)$ *and gap* ε, *then 3SAT also has a 2IP verifier with answer size c and gap* $\varepsilon \cdot \tau$.

Both lemmas above offer (pretty severe) reductions in answer sizes. Below, we show how they suffice to get the PCP theorem. Of course, the technical complexity is all hidden in the proofs of the two lemmas, which we will not be able to present. We simply mention that these lemmas are obtained by revisiting several popular "algebraic error-correcting codes" and showing that they admit query efficient probabilistic algorithms for "error-detection" and "error-correction." The reader is referred to the original papers[3, 4] for further details.

The literature on PCPs is rich with a diversity of parameters, but we chose to focus on only two: the query complexity and the gap.

PROOF OF THEOREM 2.2. We start by noting that the classical (deterministic) verifier for 3SAT is also a 2IP verifier with answer size n and gap 1. Applying Lemma 4.3 we then get it thus has a 2IP verifier with answer size $(\log n)^2$ and gap δ. Applying Lemma 4.3 again we now see that is also has a 2IP verifier with answer size $(\log(\log n))^2$ and gap δ^2. Since $a(n) = o(\log n)$ we can now apply Lemma 4.4 to see that it has a 2IP verifier with answer size c and gap $\delta^2 \cdot \tau$. By Proposition 4.2 we conclude that 3SAT has a PCP verifier with query complexity $2c$ and gap $\delta^2\tau$.

Amplifying error. We now turn to the new, arguably simpler, proof due to Dinur[12] of the PCP theorem. Since we are hiding most of the details behind some of the technical lemmas, we would not be able to completely clarify the simplicity of Dinur's approach. However, we will be able to at least show how it differs right from the top level.

Dinur's approach to the PCP theorem is an iterative one, and rather than working with large answer sizes, this proof works with small gaps (during intermediate stages).

The approach fixes a "generalized graph k-coloring" problem as the problem of interest and fixes a canonical 2IP verifier for this problem. It starts by observing that 3SAT can be transformed to this generalized graph 3-coloring problem. It then iteratively transforms this graph into a different one, each time increasing the "gap of the instance." The final instance ends up being one that where the canonical 2IP verifier either accepts with probability 1, or rejects with constant probability (depending on whether the original instance is satisfiable or not), which is sufficient for the PCP theorem. We go into some more details of this approach below, before getting into the heart of the process which is the single iteration.

A *generalized graph k-coloring* problem has as an instance a graph $G = (V, E)$ and constraint functions $\pi_e : \{1,\ldots,k\} \times \{1,\ldots,k\} \to \{\texttt{accept}, \texttt{reject}\}$ for every edge $e \in E$. The *canonical 2IP verifier* for in instance expects as provers two oracles giving $\chi_1, \chi_2 : V \to \{1,\ldots, k\}$ and does one of the following: With probability 1/2 it picks a random edge $e = (u, v) \in E$ queries for $\chi_1(u)$ and $\chi_2(v)$ and accepts iff $\pi_e(\chi_1(u), \chi_2(v)) = \texttt{accept}$. With probability 1/2 it picks a random vertex $u \in V$ and queries for $\chi_1(u)$ and $\chi_2(u)$ and accepts if and only if $\chi_1(u) = \chi_2(v)$. Note that the canonical 2IP verifier has answer size $[\log_2 k]$. An instance is *satisfiable* if the canonical 2IP verifier accepts with probability 1. An instance is *ε-unsatisfiable* if the probability that the verifier rejects is at least ε.

Key to Dinur's iterations are transformations among generalized graph coloring problems that play with the gap and the answer size (that is, # of colors allowed) of the 2IP verifiers. Since these transformations are applied many times to some fixed starting instance, it is important that the transformations do not increase the problem size by much and Dinur insists that they only increase them by a linear factor. We define this notion formally here.

Definition 4.5. *A transformation T that maps instances of generalized graph k-coloring to generalized graph K-coloring is a $(k, K, \beta, \varepsilon_0)$-linear-transformation if it satisfies the following properties:*

- *$T(G)$ has size linear in the size of G.*
- *$T(G)$ is satisfiable if G is satisfiable.*
- *$T(G)$ is $\min\{\beta\cdot\varepsilon, \varepsilon_0\}$-unsatisfiable if G is ε-unsatisfiable.*

Note that the parameter β above may be greater than 1 or smaller; and the effect in the two cases is quite different. If $\beta > 1$ then the transformation increases the gap, while if $\beta < 1$ then the transformation reduces the gap. As we will see below, Dinur's internal lemmas play with effects of both kinds (combining them in a very clever way).

The key lemma in Dinur's iterations does not play with the answer size and simply increase the gap and may be stated as below.

Lemma 4.6 (Gap Amplification Lemma). *There exists a constant $\varepsilon_0 > 0$ such that there exists a polynomial-time computable $(3, 3, 2, \varepsilon_0)$-linear-transformation T.*

Before getting a little into the details of the proof of this lemma, let us note that it suffices to prove the PCP theorem.

Proof of Theorem 2.2. We describe a 2IP verifier for 3SAT. The verifier acts as follows. Given a 3CNF formula ϕ of length n, it first applies the standard reduction from 3SAT to 3-coloring to get an instance G_0 of (generalized) graph 3-coloring which is 3-colorable iff ϕ is satisfiable. Note that this instance is $1/m$-unsatisfiable for some $m = O(n)$. The verifier then iteratively applies the transformation T to G_0 $\ell = \log m$ times, that is, it sets $G_i = T(G_{i-1})$ for $i = 1,\ldots,\ell$. Finally it simulates the canonical 2IP verifier on input G_ℓ.

If ϕ is satisfiable, then so is G_i for every i, and so the canonical 2IP verifier accepts with probability 1. If ϕ is unsatisfiable then G_i is $\min\{2^i \cdot 1/m, \varepsilon_0\}$-unsatisfiable and so G_ℓ is ε_0-unsatisfiable. Finally note that since each iteration increases the size of G_i only by a constant factor, the final graph G_ℓ is only polynomially larger than ϕ, and the entire process only requires polynomial time.

Note that the 2IP verifier thus constructed has answer size $[\log_2 3] = 2$ bits. Its gap is ε_0. The conversion to a PCP verifier leads to one that has query complexity of 4 bits and gap $\varepsilon_0 > 0$.

We now turn to the magical gap-amplifying lemma above. Dinur achieves this lemma with two sub-lemmas, where the game between answer size and gap becomes clear.

Lemma 4.7 (Gap-Increase). *For every k, $\beta_1 < \infty$, there exists a constant $K < \infty$ and $\varepsilon_1 > 0$ such that a $(k, K, \beta_1, \varepsilon_1)$-linear-transformation T_1 exists.*

Note that a large $\beta_1 \gg 1$ implies the transformation enhances the unsatisfiability of an instance. The Gap-Increase lemma is claiming that one can enhance this unsatisfiability by any constant factor for an appropriate price in the answer size. The next lemma trades off in the other direction, but with a clever and critical switch of quantifiers.

Lemma 4.8 (Answer-Reduction). *For every k there exists a constant $\beta_2 > 0$ such that for every $K < \infty$ a $(K, k, \beta_2, 1)$-linear-transformation T_2 exists.*

The constant β_2 obtained from the lemma is quite small (very close to 0). But for this price in gap-reduction we can go from large answer sizes to small ones, and the price we pay in the unsatisfiability is independent of K! This allows us to combine the two lemmas to get the powerful gap amplification lemma as follows.

Proof of Lemma 4.6. Fix $k = 3$. Let β_2 and T_2 be as in Lemma 4.8. Apply Lemma 4.7 with $\beta_1 = 2/\beta_2$ and let K, ε_1 and T_1 be as guaranteed by Lemma 4.7. Let $T(G) = T_2(T_1(G))$. Then, it can be verified that T is a $(k, k, 2, \beta_2\cdot\varepsilon_1)$-linear-transformation.

Finally we comment on the proofs of Lemmas 4.7 and 4.8. We start with the latter. The crux here is the independence of β_2 and K. A reader who attempts to use standard reductions, from say k-coloring to K-coloring would realize that this is nontrivial to achieve. But if one were to ignore the linear-size restriction, the PCP literature already gave such transformations before. In particular Lemma 4.4 gives such a transformation provided $K = 2^{o(\log n)}$. When specialized to the case $K = O(1)$ the reduction also turns out to be a linear one.

Lemma 4.7 is totally novel in Dinur's works.[12, 13] To get a sense of this lemma, let us note that its principal goal is to reduce the error of the 2IP verifier and so is related to the standard question in the context of randomized algorithms: that of error-reduction. In the context of randomized algorithms this is well studied. If one starts with any randomized algorithm to compute some function and say it produces the right answer with probability 2/3 (and errs with probability 1/3), then one can reduce the error by running this algorithm many times and outputting the most commonly seen answer. Repetition m times reduces the error to $2^{-\Omega(m)}$. One could view a 2IP verifier as just another randomized procedure and attempt to repeat the actions of the verifier m times to reduce its error. This leads to two problems. First the natural approach increases the number of rounds of communication between the verifier and the prover to m-rounds and this is not allowed (by our definitions, which were crucial to the complementary lemma). A less natural, and somewhat optimistic, approach would be to repeat the random coin tosses 2IP verifier m times, collect all the questions that it would like to ask, say, the first prover and send them together in one batch (and similarly with the second prover). Analysis of such parallel repetition of 2IP verifiers was known to be a nontrivial problem,[16, 32] yet even such an analysis would only solve the first of the problems with the "naive" approach to error-reduction. The second problem is that the transformation does not keep the size of the transformed instance linear in size and this turns out to be a fatal. Dinur manages to overcome this barrier by borrowing ideas from "recycling" of randomness,[10, 23] which suggests approaches for saving on this blowup of the instance size. Analyzing these approaches is nontrivial, but Dinur manages to do so, with a relatively clean (and even reasonably short) proof. The reader is pointed to the original paper[12] for full details, and to a more detailed survey[31] for information on the context.

Conclusion

The goal of this article is to mainly highlight the notion of a PCP, and its utility in computational complexity. Due to limitations on space and time, we were barely able to scratch the surface. In particular we did not focus on the explicit parameters and the tightest results known. The literature on PCPs is rich with a diversity of parameters, but we chose to focus on only two: the query complexity and the gap. The trade-off between the two is already interesting to study and we mention one tight version, which is extremely useful in "inapproximability" results. Håstad[22] shows that the query complexity in the PCP theorem can be reduced to 3 bits, while achieving a gap arbitrarily close to 1/2. So a verifier confronted with a fallacious assertion can read just 3 bits of the proof, and would find an error with probability (almost) one-half!

One of the somewhat strange aspects of PCP research has been that even though the existence of PCPs seems to be a "positive" statement (verification can be very efficient), its use is mostly negative (to rule out approximation algorithms). One may wonder why the positive aspect has not found a use. We suggest that positive uses might emerge as more and more of our life turns digital, and we start worrying not only about the integrity of the data, but some of the properties they satisfy, that is, we may not only wish to store some sequence of bits x, but also preserve the information that $P(x) = y$ for some program P that took x as an input.

One barrier to such uses is the current size of PCPs. PCP proofs, even though they are only polynomially larger than classical proofs; they are much larger, and this can be a prohibitive cost in practice. The good news is that this parameter is also improving. An optimistic estimate of the size of the PCP proof in the work of Håstad[22] might be around n^{10^6}, where n is the size of the classical proof! But recent results have improved this dramatically since and current best proofs[9, 12] work with PCPs of size around $O(n(\log n)^{O(1)})$ (so the constant in the exponent has dropped from 10^6 to $1 + o(1)$). Thus far, this reduction in PCP size has come at the price of increased query complexity, but this concern is being looked into by current research and so a positive use of PCPs may well be seen in the near future.

Acknowledgments

I would like to thank the anonymous reviewers for their valuable comments, and for detecting (mathematical!) errors in the earlier version of this manuscript (despite the fact that this article is not written in the PCP format).

References

1. Arora, S., Babai, L., Stern, J., Sweedyk, Z. The hardness of approximate optima in lattices, codes and systems of linear equations. *J. Comput. Syst. Sci. 54*, 2 (Apr. 1997), 317–331.
2. Arora, S., Lund, C. Hardness of approximations. *Approximation Algorithms for NP-Hard Problems*. D.S. Hochbaum, ed. PWS, 1995.
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M. Proof verification and the hardness of approximation problems. *J. ACM 45*, 3 (May 1998), 501–555.
4. Arora, S., Safra, S. Probabilistic checking of proofs: A new characterization of NP. *J. ACM 45*, 1 (Jan. 1998), 70–122.
5. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M. Checking computations in polylogarithmic time. In *Proceedings of the 23rd ACM Symposium on the Theory of Computing* (New York, 1991), ACM, 21–32.
6. Babai, L., Fortnow, L., Lund, C. Non-deterministic exponential time has two-prover interactive protocols. *Comput. Complexity 1*, 1 (1991), 3–40.
7. Babai, L., Moran, S. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity class. *J. Comput. Syst. Sci. 36*, 2 (Apr. 1988), 254–276.
8. Ben-Or, M., Goldwasser, S., Kilian, J., Wigderson, A. Multi-prover interactive proofs: How to remove intractability. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing* (1988), 113–131.
9. Ben-Sasson, E., Sudan, M. Short PCPs with poly-log rate and query complexity. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing* (New York, 2005), ACM, 266–275.
10. Cohen, A., Wigderson, A. Dispersers, deterministic amplification, and weak random sources (extended abstract). In *IEEE Symposium on Foundations of Computer Science* (1989), 14–19.
11. Cook, S.A. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium Theory of Computing* (Ohio, 1971), Shaker Heights, 151–158.
12. Dinur, I. The PCP theorem by gap amplification. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing* (New York, 2006), ACM, 241–250. Preliminary version appeared as an ECCC Technical Report TR05-046.
13. Dinur, I., Reingold, O. Assignment testers: Towards a combinatorial proof of the PCP-theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science* (Loc Alamitos, CA, USA, 2004), IEEE, 155–164.
14. Feige, U. A threshold of ln *n* for approximating set cover. *J. ACM 45*, 4 (1998), 634–652.
15. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M. Interactive proofs and the hardness of approximating cliques. *J. ACM 43*, 2 (1996), 268–292.
16. Fortnow, L., Rompel, J., Sipser, M. On the power of multi-prover interactive protocols. *Theor. Comput. Sci. 134*, 2 (1994), 545–557.
17. Garey, M.A., Johnson, D.S. *Computers and Intractability*. Freeman, 1979.
18. Goldreich, O. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1998.
19. Goldreich, O., Micali, S., Wigderson, A. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM 38*, 1 (July 1991), 691–729. Preliminary version in *IEEE FOCS*, 1986.
20. Goldwasser, S., Micali, S., Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput. 18*, 1 (February 1989), 186–208.
21. Håstad, J. Clique is hard to approximate within n to the power 1-epsilon. *Acta Mathemat. 182* (1999), 105–142.
22. Håstad, J. Some optimal inapproximability results. *J. ACM 48* (2001),798–859.
23. Impagliazzo, R., Zuckerman, D. How to recycle random bits. In *IEEE Symposium on Foundations of Computer Science* (1989), 248–253.
24. Karloff, H., Zwick, U. A 7/8-approximation algorithm for max 3sat? In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science* (*FOCS '97*) (Washington, DC, USA, 1997), IEEE Computer Society, 406–415.
25. Karp, R.M. Reducibility among combinatorial problems. *Complexity of Computer Computations*. R. Miller, and J. Thatcher, eds.1972, 85–103.
26. Khot, S. Guest column: Inapproximability results via long code based pcps. *SIGACT News 36*, 2, 25–42, 2005.
27. Levin, L.A. Universal search problems. *Problemy Peredachi Informatsii*, *9*, 3 (1973), 265–266.
28. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N. Algebraic methods for interactive proof systems. *J. ACM 39*, 4 (Oct. 1992), 859–868.
29. Lund, C., Yannakakis, M. On the hardness of approximating minimization problems. *J. ACM 41*, 5 (Sept. 1994), 960–981.
30. Papadimitriou, C., Yannakakis, M. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci. 43* (1991), 425–440.
31. Radhakrishnan, J., Sudan, M. On Dinur's proof of the PCP theorem. *Bulletin (New Series) Amer. Math. Soc.*, *44*, 1 (Jan. 2007), 19–61.
32. Raz, R. A parallel repetition theorem. *SIAM J. Comput. 27*, 3 (1998), 763–803.
33. Shamir, A. IP = PSPACE. *J. ACM 39*, 4 (Oct. 1992), 869–877.
34. Shannon, C.E. A mathematical theory of communication. *Bell Syst. Tech. J. 27*, (1948), 379–423, 623–656.

Madhu Sudan is Fujitsu Professor of EECS at MIT's Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.

DOI:10.1145/1467247.1467268

Technical Perspective The Beauty of Error-Correcting Codes

By Daniel A. Spielman

ERROR-CORRECTING CODES ARE the means by which we compensate for interference in communication, and are essential for the accurate transmission and storage of digital data. All communication mechanisms and storage devices are subject to interference, typically called "noise," which corrupts communicated messages and stored data. Thus, for a communication system to faithfully transmit data, it must build redundancy into its transmissions in such a way that even if a transmission is partially corrupted, the intended message may be reconstructed. Error-correcting codes provide the mapping from messages to redundant transmissions.

For example, a message is usually a string of zeros and ones. A redundant encoding of a message may be obtained by appending a few parity bits to the original message, to form a *codeword*. The *rate* of a code is the ratio of the length of a message to the length of a codeword, and equals the reciprocal of the redundancy. A communication medium, called a channel, might transmit bits, and noise could flip bits from zero to one or one to zero. For example, the Binary Symmetric Channel with crossover probability p transmits bits, and flips each bit with probability p, independently. An error-correcting code is designed with an abstract model of the target communication channel in mind.

Given a model of a channel, one should design a code that maximizes the rate while minimizing some tradeoff of error-probability, delay, and the computational complexity of encoding and decoding. While the goal of achieving low probability of error in a communication system is fundamentally probabilistic, major advances in the field have been made through a worst-case, deterministic, approach. The paper here by Guruswami and Rudra surveys developments in the worst-case approach to the coding problem, and explains their own recent contributions. They build on the classical Reed-Solomon codes.

Reed-Solomon codes employ a signaling alphabet containing more elements than just zero and one: each symbol is an element of a finite field, such as the integers modulo a prime. In a Reed-Solomon code of rate R, classic decoding algorithms can efficiently reconstruct a message so long as at most a $(1–R)/2$ fraction of the symbols in the transmitted codeword are corrupted. This is exactly the fraction of errors up to which the problem is guaranteed to have a unique solution: there exist rare patterns containing just one more error for which two codewords are equally close to the corrupted transmission.

A major advance in the decoding of Reed-Solomon codes was Sudan's[4] algorithm for list decoding Reed-Solomon codes. A list-decoding decoder returns the list of all codewords within some distance of a corrupted transmission. While the closest codeword is usually unique, the algorithmic task is simplified by the option of returning a list. Guruswami and Sudan's[1] improvement of Sudan's list decoder efficiently returns the list of all codewords that differ from a corrupted transmission in at most a $1 - \sqrt{R}$ fraction of symbols, and the list is guaranteed to be short.

This was a big improvement over previous decoding algorithms, but made little difference at the desirable high rates (near 1), where $1 - \sqrt{R}$ is approximately the same as $(1–R)/2$. Guruswami and Rudra's advance exploits an idea of Parvaresh and Vardy[3] for bundling Reed-Solomon alphabet symbols together. This makes the signaling alphabet slightly larger, but greatly increases the fraction of errors under which efficient list decoding is possible. They obtain codes of rate R from which one can efficiently produce the list of all codewords that differ from a corrupted transmission in a fraction of symbols approaching $1 - R$. For high-rate codes, this is almost twice as many errors as previous schemes could correct. Moreover, we know that one cannot hope to do better.

While a tremendous theoretical advance, more work is required before these codes can be used in practical communication systems. The decoding algorithms run in polynomial time, but need to be faster before they can be applied in practice. They also need to be extended to incorporate information from lower levels of the communication system. Few communication media naturally transmit finite field elements, or even zeros and ones. These symbols are usually converted into analog waveforms. Receivers of partially corrupted waveforms can do more than just report which valid waveform is closest: they can return the likelihood of each valid waveform. A *soft-decision decoder* incorporates this information into the decoding process.

Koetter and Vardy[2] figured out how to incorporate such information in the Guruswami-Sudan algorithm, and an analogous discovery may be required before we communicate using Guruswami-Rudra codes. C

References

1. Guruswami, V. and Sudan, M. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Trans. on Info. Theory*, *45* (1999), 1757–1767.
2. Koetter, R. and Vardy, A. Algebraic soft-decision decoding of Reed-Solomon codes. *IEEE Trans. on Info. Theory 49*, 112 (2003), 2809–2825.
3. Parvaresh, F. and Vardy, A. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science* (2005), 285–294.
4. Sudan, M. Decoding the Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity 13*, 1 (1997), 180–193.

Daniel A. Spielman (Spielman@cs.yale.edu) is a professor of applied mathematics and computer science at Yale University, New Haven, CT.

DOI:10.1145/1467247.1467269

Error Correction up to the Information-Theoretic Limit

By Venkatesan Guruswami and Atri Rudra

Abstract

Ever since the birth of coding theory almost 60 years ago, researchers have been pursuing the elusive goal of constructing the "best codes," whose encoding introduces the minimum possible redundancy for the level of noise they can correct. In this article, we survey recent progress in *list decoding* that has led to *efficient* error-correction schemes with an *optimal* amount of redundancy, even against *worst-case errors* caused by a potentially malicious channel. To correct a proportion ρ (say 20%) of worst-case errors, these codes only need close to a proportion ρ of redundant symbols. The redundancy cannot possibly be any lower information theoretically. This new method holds the promise of correcting a factor of two more errors compared to the conventional algorithms currently in use in diverse everyday applications.

1. INTRODUCTION

Coping with corrupt data is an essential part of our modern day lives; even if most of the time we are blissfully unaware of it. When we watch television, the TV has to deal with signals that get distorted during transmission. While talking on our cellphones, the phone has to work with audio signals that get corrupted during transmission through the atmosphere though we definitely are aware of it when the connection is poor. When we surf the Internet, the TCP/IP protocol has to account for packets that get lost or garbled while being routed. When we watch movies on DVDs, the player has to overcome loss of data due to scratches. Even when we buy our groceries, the scanner has to deal with distorted barcodes on packages.

The key ingredient in coping with errors in these and many other applications is an *error-correcting code* or just *code* for brevity. The idea behind codes is conceptually simple: add redundancy to the information so that even if the resulting data gets corrupted, e.g. packets get corrupted during routing or the DVD gets some scratches, the original information can still be recovered.

Ideally, we would like to add a small amount of redundancy and at the same time be able to correct many errors. As one might expect, these are conflicting goals and striking the right balance is where things get interesting. For example, consider the code where every information bit is repeated say a 100 times (this is known as the repetition code). Intuitively, this code should do well. In particular, the following is a natural error-recovery procedure or a *decoding algorithm*: for every consecutive 100 bits of the data, identify whether the majority of the bits is 0 or 1, and output the corresponding bit. Unless we happen to be unlucky, this decoding algorithm can recover from quite a few errors. The downside is that every 100 bits of data contain only *one* bit of information—imagine how large a DVD would need to be in order to store a movie with such redundancy. On the other extreme is the parity code, which appends the parity of the information bits at the end of the message. This code uses the minimum amount of redundancy possible but has poor error-recovery capabilities. Indeed, even if just two bits get flipped, it can go undetected. For example, 0001 gets encoded as 00011 under the parity code. If the first two bits get corrupted and we receive 11011, we would misinterpret the original message to be 1101. Imagine Clark Gable saying at the end of your small parity encoded DVD for *Gone with the Wind*, "Frankly, my dear, I don't give a ham !"

To capture this inherent tension between the redundancy and the error tolerance of codes, let us define codes and some key parameters formally. A code C is given by an *encoding* map of the form $C : \Sigma^k \rightarrow \Sigma^n$ (for integers $k < n$) which encodes a sequence of k symbols from Σ into a larger sequence of n symbols. Given a *message* $m \in \Sigma^k$, $C(m)$ is known as the corresponding *codeword*. The parameters k, n, and Σ are called the *dimension*, *block length*, and *alphabet* of C, respectively. We will often use the ratio $R = k/n$, which is called the *rate* of C. Note that R exactly captures the amount of information contained per bit of a codeword. The *Hamming distance* between two strings in Σ^n is the number of coordinates where they differ. The *minimum distance* of a code is defined to be the smallest Hamming distance between two distinct codewords.

Thus, our question of interest can be now re-stated as follows: given a code C of rate R, what is the maximum fraction of errors (which we will henceforth refer to as ρ) that can be tolerated by C? Now as every codeword has k symbols of information, it is intuitive that in the worst case at least k symbols of a codeword should be uncorrupted to have any hope of recovering the original information. In other words, we can only have $\rho \le (n - k)/n = 1 - R$, irrespective of the computational power of the decoder.

The main focus of this article is the following question:

> Can we construct a code C of rate R that can be efficiently decoded from close to a $1 - R$ fraction of errors?

Quite surprisingly, we will show the answer to be yes. Thus, the above simple information-theoretic limit can in fact be approached. In particular, for small rates, we can recover from situations where almost all of the data can be corrupted. For example, we will be able to recover even if Clark Gable were to spout "alhfksa, hy meap xH don'z hive b hayn!" There are

The original version of this paper was published in *IEEE Transactions on Information Theory*, 2008.

in fact two parts to the above question. First, the code should be such that the identity of the message is uniquely (or near uniquely) pinned down based on the noisy version of its encoding. Second, we need an *efficient* procedure to recover the message based on the corrupted codeword, with runtime bounded by a hopefully small polynomial in the block length n. Brute-force approaches such as searching through all codewords require time exponential in n, and are computationally prohibitive. The main message of this article is that a variant of the widely deployed Reed–Solomon codes can in fact be error-corrected *efficiently*, even as the fraction of errors approaches the absolute $1 - R$ limit.

We stress that in the above question, the noise model is a *worst-case* one, where the channel is modeled as an adversary/jammer that can corrupt the codeword in an arbitrary manner, subject only to a limit on the total number of errors. No assumptions are made on how the errors are distributed. The worst-case model was put forth in Hamming's influential paper.[12] An alternate approach, pioneered by Shannon in his seminal work,[16] is to model the noisy channel as a stochastic process whose behavior is governed by a precisely known probability law. A simple example is the binary symmetric channel (BSC_ρ) where each bit transmitted gets flipped with a probability ρ, independent of all other bits. For every such channel, Shannon exactly characterized the largest rate at which reliable communication is possible.

We note that obtaining *algorithmic* results is more difficult in worst-case noise model. In fact, traditionally it was believed that codes designed for worst-case noise faced a limit of $(1 - R)/2$ fraction of errors, which is factor 2 off from the information-theoretic limit of a $(1 - R)$ fraction of errors. In attempting to correct $e > (n - k)/2$ errors, we face a problem: there may be more than one codeword within Hamming distance e from the received word. In any code mapping k symbols to n symbols, there must be two codewords at distance $(n - k + 1)$ or less. If one of these codewords is transmitted and gets distorted by errors to halfway between the two codewords, unambiguous recovery of the original message becomes infeasible. This suggests that beyond a fraction $(1 - R)/2$ of worst-case errors, the original message is unrecoverable. This was indeed the conventional wisdom in coding theory till the 1990s.

A natural strategy, in the event of multiple close-by codewords, would be to allow the decoder to output a list of codewords. This model is called *list decoding*. It was introduced in the late 1950s by Elias[2] and Wozencraft,[19] and revived with an algorithmic focus for a cryptographic application by Goldreich and Levin.[4] Further, it turns out that the bad case above is rather pathological—for typical patterns of e errors, for e much bigger than $(n - k)/2$, the original codeword will be the only codeword within distance e. Thus, list decoding in addition to handling the bad error patterns for "unique" decoding, also allows us to uniquely recover the transmitted codeword for *most* error patterns.

It is interesting to note that even though the list decoding problem has a long history in the coding theory world,[a] a large share of the algorithmic progress in list decoding has happened in the theoretical computer science community. One of the reasons for this happy coincidence is that list decoding has found numerous applications in cryptography and computational complexity theory (e.g., see the discussion on randomness extractors in Section 5).

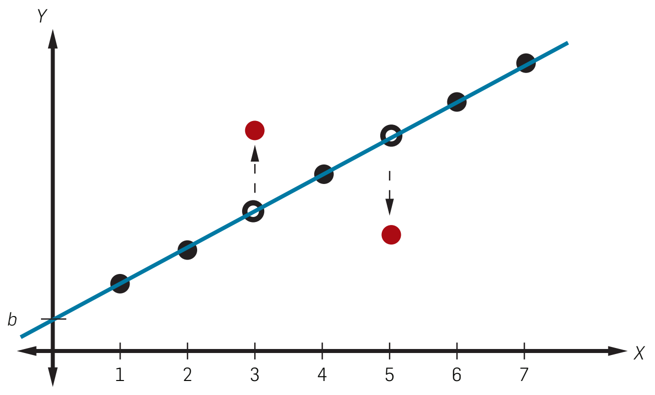
In particular, in the last decade, the subject of list decoding has witnessed a lot of activity, culminating in algorithms that correct close to the information-theoretically optimal $1 - R$ fraction of errors with rate R. The purpose of this article is to discuss this recent body of results which deliver the full promise of codes against worst-case errors. We begin in Section 2 by describing a popular family of codes and a few decoding algorithms for it.

2. REED–SOLOMON CODES

In 1960, Irving Reed and Gustave Solomon described a construction of error-correcting codes, which are called Reed–Solomon codes after them, based on polynomials over finite fields.[b] Almost 50 years after their invention, Reed–Solomon codes (henceforth, RS codes) remain ubiquitous today in diverse applications ranging from magnetic recording to UPS bar codes to satellite communications. To describe the simple and elegant idea behind RS codes, imagine Alice wishes to communicate a pair of numbers (a, b) to Bob. We can think of (a, b) as specifying a line in the plane (with X, Y axes) with equation $Y = aX + b$. Clearly, to specify the line, it suffices to communicate just two points on the line. To guard against errors, Alice can oversample this line and send more points to Bob, so that even if a few points are distorted by errors, the collection of points resembles the original line more closely than any other line (Figure 1). Thus, Bob can hope to recover the correct line, and in particular (a, b).

To encode longer messages consisting of k symbols via an RS code, one thinks of these as the coefficients of a polynomial $f(X)$ of degree $k - 1$ in a natural way, and encodes the message as $n > k$ points from the curve $Y - f(X) = 0$.

Figure 1: Oversampling from a line $Y = aX + b$ to tolerate errors, which occur at $X = 3$ and 5.



[a] The problem was introduced about 50 years ago and the main combinatorial limitations of list decoding were established in the 1970s and 1980s.

[b] For this article, readers not conversant with fields can think of a finite field as $\{0, 1, \dots, p-1\}$ for a prime p with addition and multiplication operations defined modulo p.

Equivalently, the encoding consists of the values of the polynomial $f(X)$ at n different points. Formally, if $\mathbb{F}$ is a finite field with at least n elements, and $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ is a sequence of n *distinct* elements, the RS encoding $\mathrm{RS}_{\mathbb{F},S,n,k}(m)$ of a message $m = (m_0, m_1, \ldots, m_{k-1})$ is given by

$$\mathrm{RS}_{\mathbb{F},s,n,k}(m) = (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_3))$$

where $f(X) = m_0 + m_1X + \ldots + m_{k-1}X^{k-1}$. We stress here that the choice of S is up to the code designer—we will exploit this feature in Section 3.2.

The following basic algebraic fact will be crucial:

> A non-zero polynomial $p(X)$ of degree D over a field $\mathbb{F}$ can have at most D distinct roots, i.e., at most D elements $\alpha \in \mathbb{F}$ can satisfy $p(\alpha) = 0$.

This fact implies that the encodings of two distinct messages m and m' by $\mathrm{RS}_{\mathbb{F},S,n,k}$ must differ in more than $n - k$ locations.[c] The minimum distance of the RS code is thus at least $n - k + 1$. It is in fact equal to $n - k + 1$: e.g., consider encodings of the messages corresponding to the zero polynomial and the polynomial $\prod_{i=1}^{k-1}(X - \alpha_i)$. A minimum distance of $(n - k + 1)$ is the best possible for a code of dimension k, making RS codes optimal in this regard.

2.1. Correcting errors in RS codewords

Why is the above large distance useful? If at most $(n - k)/2$ errors corrupt the evaluations of a polynomial $f(X)$, then the encoding of $f(X)$ remains the best fit of the corrupted data in terms of agreements than the encoding of any other polynomial $g(X)$ of degree less than k. Thus, one can hope to recover $f(X)$ and the correct message even in the presence of $(n - k)/2$ errors. However, it is not immediate how to isolate the errors and recover $f(X)$ *efficiently*. We do not know the locations of the errors, and trying all possibilities will need exponential time.

Back in 1960, even before polynomial running time was formalized as the notion underlying efficient algorithms, Peterson[15] described a polynomial time algorithm to solve the above problem. We now describe the high-level idea behind a different algorithm, due to Welch and Berlekamp,[18] following the elegant description in Gemmell and Sudan.[3]

Assume that the encoding $(f(\alpha_1), \ldots, f(\alpha_n))$ of a polynomial $f(X)$ was transmitted, and we receive a corrupted version $(y_1, y_2, \ldots, y_n)$, where the set $E = \{i : y_i \neq f(\alpha_i)\}$ of error locations has size at most $(n - k)/2$. Suppose we miraculously *knew* the set E. Then we could simply discard the y_i's corresponding to these locations, and interpolate $f(X)$ through the rest of the correct data points. We will have at least $(n + k)/2 \geq k$ locations, so interpolation will uniquely identify $f(X)$.

Error Location via Bivariate Interpolation: The key is thus a clever method to locate the set E of error locations quickly. To motivate this, let us cast the problem geometrically as an equivalent *noisy curve fitting* problem. We are given n points (α_i, y_i), $i = 1, 2, \ldots, n$, in the "plane" $\mathbb{F} \times \mathbb{F}$. The goal is to find the unique curve with equation $Y - f(X) = 0$ where degree$(f) < k$ that passes through all but $e \leq (n - k)/2$ locations i, namely those in E. If there was no noise, fitting a curve through all n points is easy—it is just polynomial interpolation. We know $Y - f(X)$ passes through $n - e$ points, so we can get a curve that passes through all the points by fitting vertical lines through the error points along with the curve $Y - f(X) = 0$; see Figure 2. Algebraically, if we define

$$P(X, Y) = (Y - f(X)) \prod_{i \in E} (X - \alpha_i), \tag{1}$$

then the curve $P(X, Y) = 0$ passes through all the points, i.e., $P(\alpha_i, y_i) = 0$ for every i. The second factor in the expression (1) is called the error-locator polynomial, which has the error locations as its roots.

Given $P(X, Y)$, one can find its factors (which can be done efficiently) and thus read off the message polynomial $f(X)$ from the $Y - f(X)$ factor. But how do we find $P(X, Y)$? Finding $P(X, Y)$ in its factored form (1) is begging the question, but note that we can also write $P(X, Y)$ in the form $P(X, Y) = D_1(X)Y - N_1(X)$ where degree$(D_1) \leq e \leq (n - k)/2$ and degree$(N_1) < e + k \leq (n + k)/2$.

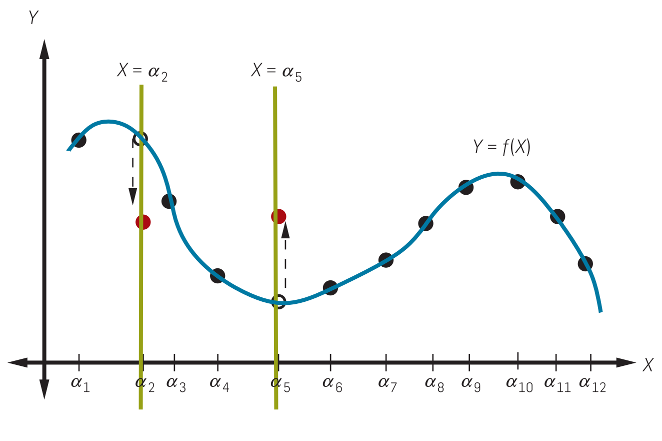
Knowing such a polynomial exists, we can try to find a nonzero bivariate polynomial $Q(X, Y) = D_2(X)Y - N_2(X)$ satisfying

1. degree$(D_2) \leq (n - k)/2$ and degree $(N_2) < (n + k)/2$
2. $Q(\alpha_i, y_i) = 0$ for $i = 1, 2, \ldots, n$

This can be done by setting up a system of linear equations over $\mathbb{F}$ with unknowns being the coefficients of $D_2(X)$ and $N_2(X)$, and n linear constraints $Q(\alpha_i, y_i) = 0$. We have called the polynomial $Q(X, Y)$ since we cannot assume that the solution will in fact equal $P(X, Y)$ (there may be multiple nonzero solutions to the above system). Solving this linear system can certainly be done in polynomial time, and also admits fast, practical methods.

One can prove, using elementary algebra, that when the number of errors $e \leq (n - k)/2$, *any* interpolated $Q(X, Y)$ satisfying the above two conditions *must* have $P(X, Y)$ as a factor,

Figure 2: An RS codeword (polynomial $f(X)$ evaluated on points $\alpha_1, \alpha_2, \ldots, \alpha_{11}$); its corruption by two errors (at locations α_2 and α_5); and illustration of the curve fitting through the noisy data using correct curve and "error-locator lines."



[c] If not, $\mathrm{RS}_{\mathbb{F},S,n,k}(m) - \mathrm{RS}_{\mathbb{F},S,n,k}(m')$, which corresponds to the evaluation of the non-zero polynomial $\sum_{i=0}^{k-1}(m_i - m_i')X^i$ of degree at most $k - 1$, has at least k zeroes: a contradiction.

and be of the form $P(X, Y)\, A(X)$ for some nonzero (possibly constant) polynomial $A(X)$. The intuitive reason is that since there are so few errors in the data compared to the curve $Y - f(X) = 0$, the curve $P(X,Y) = 0$ is the most economical way to fit all the data points. The formal proof proceeds by considering the polynomial $R(X) \overset{\text{def}}{=} Q(X, f(X))$, and arguing it must be identically zero since (i) it has at least $(n + k)/2$ roots (namely all α_i's for which $f(\alpha_i) = y_i$) and (ii) it has degree less than $(n + k)/2$ (by design of $Q(X, Y)$). Thus, $Q(X, Y)$ also has $Y - f(X)$ as a factor, and we can recover $f(X)$ by factoring $Q(X, Y)$. (The actual task here is easier than general bivariate polynomial factorization, and admits near-linear time algorithms.)

2.2. List decoding Reed–Solomon codes

We now turn to list decoding of Reed–Solomon codes. The setup is as before: given n points $(\alpha_i, y_i) \in \mathbb{F}^2$, find polynomials $f(X)$ of degree less than k such that $f(\alpha_i) \neq y_i$ for at most e locations i. The difference is that now $e \gg (n - k)/2$, and so there may be more than one such polynomial $f(X)$ that the decoder needs to output.

Before delving into the algorithms, we pause to consider how large a number of errors e one can target to correct. Clearly, we need the guarantee there will be only a few (say, at most polynomially many in n) solution polynomials $f(X)$ or else there is no hope for the decoder to output all of them in polynomial time. Using the fact that encodings of any two polynomials differ in more than $(n - k)$ locations, it can be shown (via the so-called "Johnson bound") that for $e \le n - \sqrt{kn}$, the number of solutions (called the *list size*) is guaranteed to be polynomially small. Whether one can prove a polynomial list size bound for certain RS codes for even larger e remains a key open question.

We now describe the idea underlying Sudan's breakthrough algorithm for list decoding RS codes.[17] Recall that we want to solve a noisy curve fitting problem, and output all curves $Y - f(X) = 0$ with $\deg(f) < k$ that pass through $n - e$ or more of the n points (α_i, y_i). For $e \le (n - k)/2$, the Berlekamp–Welch algorithm interpolated a bivariate polynomial $Q(X, Y)$ of a very specific format through all the n points. Sudan's idea for $e > (n - k)/2$ was to interpolate/fit a *general* nonzero curve $Q(X, Y) = 0$ of just high enough "degree" (so that its existence is guaranteed) through *all* the n points. Fitting such a curve can be done efficiently by solving a system of linear equations to determine the coefficients of $Q(X, Y)$.

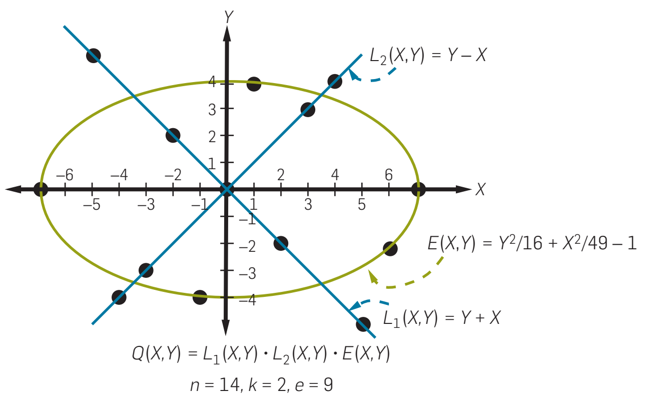
For the Berlekamp–Welch algorithm, arguing that $Y - f(X)$ was a factor of $Q(X, Y)$ followed from the very special structure of $Q(X, Y)$. In the list decoding case, Sudan exploited special properties of intersections of curves of the form $Y - f(X)$ with any interpolated bivariate polynomial $Q(X, Y)$ with appropriate degree constraints. Informally, Sudan's idea is that given the strong degree constraint on $Q(X, Y)$, every curve $Y - f(X) = 0$ with $\deg(f) < k$ that picks up at least $n - e$ of points must be "used" by the interpolated curve in meeting the requirement to pass through all n points. As an example, in Figure 3, the goal is to find all lines (i.e., we have $k = 2$) that pass through all but $e = 9$ of the $n = 14$ input points (there are two such lines, marked in the figure as $L_1(X, Y)$ and $L_2(X, Y)$). There are enough degrees of freedom in the equation of a degree 4 curve so that one can fit a degree 4 curve through any set of 14 points. The figure illustrates one such curve, which is the product of the two lines with an "ellipse" $E(X, Y)$. (Note that the total degree of $Q(X, Y)$ is 4.) Further, we see that the two relevant lines pop out as factors. This is not a coincidence, and every degree 4 curve passing through the 14 points must have these two lines as factors. The reason: if a line is not a factor, then it can intersect a degree 4 curve in at most 4 points. Since each of these lines intersects any interpolated curve in at least 5 points, it must be a factor.

More formally, the "degree" measure of the interpolated polynomial $Q(X,Y)$ will be the $(1, k - 1)$-degree, which is defined as the maximum of $i + (k - 1)j$ over all monomials X^iY^j that occur with a nonzero coefficient in $Q(X, Y)$. Let D denote the $(1, k - 1)$ degree of $Q(X, Y)$. Generalizing the above argument for lines, if a curve $Y - f(X) = 0$ with $\deg(f) < k$ passes through more than D points, then $Y - f(X)$ must be a factor of $Q(X, Y)$. With a counting argument, one can show that a $(1, k - 1)$-degree D of $\sqrt{2kn}$ suffices to fit a nonzero curve. Together, this leads to an algorithm that can correct $n - \sqrt{2kn}$ errors, or a fraction $1 - \sqrt{2R}$ of errors as a function of the rate R.

For low rates, this algorithm enables recovery even in settings when noise overwhelms correct data, and close to 100% of the symbols may be in error. This feature sets the stage for several powerful applications in cryptography and complexity theory. However, the algorithm does not give any improvement over the $(1 - R)/2$ error fraction corrected by traditional algorithms for rates $> 1/3$, and also falls short of the $1 - \sqrt{R}$ radius suggested by the combinatorial bounds.

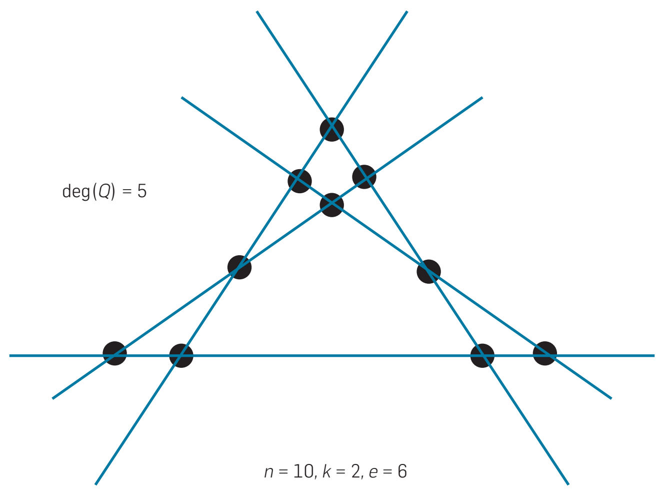
We now turn to the improved algorithm correcting a fraction $1 - \sqrt{R}$ of errors due to Guruswami and Sudan.[10] The key new idea is to insist that the interpolated polynomial $Q(X, Y)$ has *multiple* zeroes at each of the n points. To explain this, we attempt a high-level geometric description. Consider the example in Figure 4 with $n = 10$ points, the goal being to output all lines that pass through at least $n - e = 4$ points. This example cannot be solved by Sudan's algorithm. Indeed, since there are five solution lines, if they are all factors of some interpolated curve, the curve must have degree at least 5. However, there is no guarantee that an arbitrary degree 5 curve through the points must have every line passing through 4 of the points as a factor (the line has to pass

Figure 3: Illustration of list decoding of RS code that evaluates lines over the points −7, −5, −4, . . . , 4, 5, 6, 7. The two lines are recovered as factors of a degree 4 interpolated curve through all the points.



Y
X
$L_2(X,Y) = Y - X$
$E(X,Y) = Y^2/16 + X^2/49 - 1$
$L_1(X,Y) = Y + X$
$Q(X,Y) = L_1(X,Y) \cdot L_2(X,Y) \cdot E(X,Y)$
$n = 14, k = 2, e = 9$

Figure 4: Illustration of Guruswami–Sudan algorithm for list decoding RS codes. The lines are recovered as factors of a degree 5 curve that passes through each point twice.



through 6 points to guarantee this). Let C^* be the degree 5 curve that is the product of the five solution lines. As mentioned above, if we interpolate a degree 5 curve through the 10 points, we will in general *not* get C^* as the solution. However, notice a special property of C^*—it passes through each point *twice*; a priori there is no reason to expect that an interpolated curve will have this property. The Guruswami–Sudan idea is to *insist* that the interpolation stage produces a degree 5 curve with zero of multiplicity at least 2 at each point (i.e., the curve intersects each point twice). One can then argue that each of the five lines must be a factor of the curve. In fact, this will be the case for degree up to 7. This is because the number of intersections of each of these lines with the curve, counting multiplicities, is at least $4 \times 2 = 8$ which is greater than the degree of the curve. Finally, one can always fit a degree 7 curve passing through any 10 points twice (again by a counting argument). So by insisting on multiplicities in the interpolation step, one can solve this example.

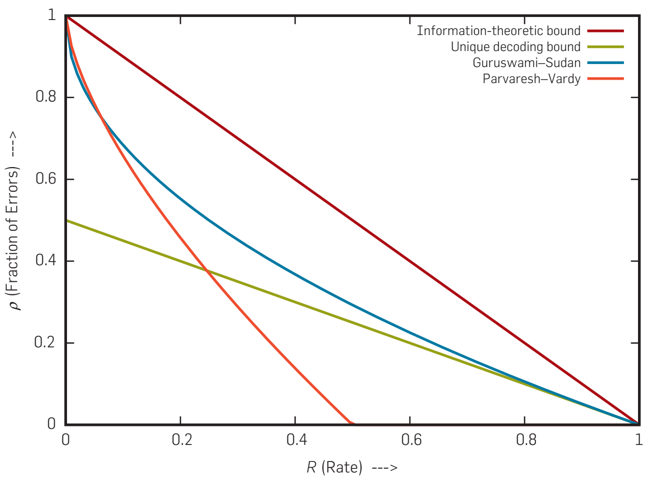
In general, the interpolation stage of the Guruswami–Sudan list decoder finds a polynomial $Q(X, Y)$ that has a zero of multiplicity w for some suitable integer w at each (α_i, y_i).[d] Of course this can always be accomplished with a $(1, k - 1)$-degree that is a factor w larger (by simply raising the earlier interpolation polynomial to the w'th power). The key gain is that the required multiplicity can be achieved with a degree only about a factor $w/\sqrt{2}$ larger. The second step remains the same, and here each correct data point counts for w zeroes. This $\sqrt{2}$ factor savings translates into the improvement of ρ from $1-\sqrt{2R}$ to $1-\sqrt{R}$. See Figure 5 for a plot of this trade-off between rate and fraction of errors, as well as the $(1 - R)/2$ trade-off of traditional unique decoding algorithms. Note that we now get an improvement for *every* rate. Also plotted are the information-theoretic limit $1 - R$, and the Parvaresh–Vardy improvement for low rates that we will discuss shortly.

Soft Decoding: We now comment on a further crucial benefit of the multiplicities idea which is relevant to potential practical applications of list decoding. The multiplicities can be used to encode the relative importance of different codeword positions, using a higher multiplicity for symbols whose value we are more confident about, and a lower multiplicity for the less reliable symbols that have lower confidence estimates. In practice, such confidence estimates (called "soft inputs") are available in abundance at the input to the decoder (e.g., from demodulation of the analog signal). This has led to a promising *soft-decision* decoder for RS codes with good coding gains in practice,[13] which was adopted in the Moonbounce program to improve communication between Ham radio operators who bounce radio signals off the moon to make long distance contacts.

3. FOLDED REED–SOLOMON CODES

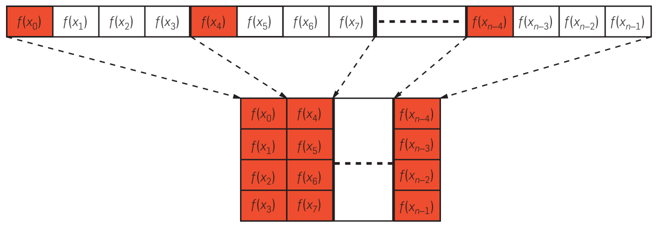
We now discuss a variant of RS codes called *folded Reed–Solomon codes* (henceforth folded RS codes), which will let us approach the optimal error-correction radius of a fraction $1 - R$ of errors. The codewords in the folded RS code will be in one-to-one correspondence with RS codewords. We begin with an informal description. Consider the RS codeword corresponding to the polynomial $f(X)$ that is evaluated at the points $x_0, x_1, \ldots, x_{n-1}$ from $\mathbb{F}$, as depicted by the codeword on top in Figure 6. The corresponding codeword in the folded RS code (with *folding parameter* of $m = 4$) is obtained by juxtaposing together four consecutive symbols on the RS codeword as shown at the bottom of Figure 6. In other words, we think of the RS code as a code over a larger alphabet (of four times the "packet size") and of block length four times smaller. This repackaging reduces the number of error patterns one has to handle. For example, if we are targeting correcting errors in up to a 1/4 fraction of the new larger symbols, then we are no longer required to correct the error pattern corresponding to the (pink) shaded columns in Figure 6 (whereas the same

Figure 5: Rate vs. error-correction radius for RS codes. The optimal trade-off is also plotted, as is the Parvaresh–Vardy's improvement over RS codes.



[d] We skip formalizing the notion of multiple zeroes in this description, but this follows along standard lines and we refer the interested reader to Guruswami and Sudan[10] for the details.

Figure 6: Folding of the Reed–Solomon code with parameter $m = 4$. Each column represents an alphabet character.



$f(x_0)$ $f(x_1)$ $f(x_2)$ $f(x_3)$ $f(x_4)$ $f(x_5)$ $f(x_6)$ $f(x_7)$ $f(x_{n-4})$ $f(x_{n-3})$ $f(x_{n-2})$ $f(x_{n-1})$

error pattern over the original symbols needs to be taken care of in the Reed–Solomon case).

We would like to stress on a subtle point here: in the worst-case error model, the "atomic" unit of error is an alphabet character. This was used crucially in the example above to rule out an error pattern that was admissible for the smaller alphabet. For the reader who might we worried that this constitutes "cheating," e.g., what if one collapses the entire RS codeword into one large symbol, we offer two counter-points. First, since we will only use a *constant* folding parameter, the increase in alphabet size from that of RS codes is modest. Second, in Section 4, we will see how to convert folded RS codes into codes over alphabets whose size *does not depend at all* on the block length, while still maintaining similar error-correction properties.

We now formally define the folded RS code. Let the nonzero elements of the field $\mathbb{F}$ be generated by γ, i.e., every nonzero element is γ^i for some $0 \le i \le |\mathbb{F}| - 2$ (such a γ always exists for any finite field $\mathbb{F}$). Let $m \ge 1$ be the *folding parameter* and let n be an integer that is divisible by m and $n \le |\mathbb{F}| - 1$. The folded RS encoding (with folding parameter m) of the message polynomial $f(X)$ has as its j'th symbol for $0 \le j < n/m$, the m-tuple $(f(\gamma^j m), f(\gamma^{jm+1}), \ldots, f(\gamma^{jm+m-1}))$.

The block length of these codes is $N = n/m$. The rate of the code remains k/n, since the folding operation does not introduce any further redundancy.

The folding operation does restrict the error patterns that one needs to worry about. But how can one actually exploit this in a decoding algorithm and manage to correct a larger fraction of errors compared to the unfolded RS codes? We turn to this question next.

3.1. Multivariate decoding

Recall the two step Guruswami–Sudan (GS) algorithm. First, we interpolate a bivariate polynomial $Q(X, Y)$ through the points $(\alpha_i, y_i) \in \mathbb{F}^2$. Then in the second step, we factorize the bivariate polynomial and retain factors of the form $Y - f(X)$, where $f(X)$ is a polynomial of degree less than k (there might be factors that are not linear in Y: we ignore them). Let us recast the second step in an equivalent description, which will be useful later. In particular, consider the *univariate* polynomial $R_X(Y)$ equivalent to $Q(X, Y)$, where the coefficients themselves are *polynomials* in indeterminate X with their own coefficients from $\mathbb{F}$: given the polynomial $Q(X, Y)$ one can compute $R_X(Y)$ by collecting all the coefficients of the same power of Y together (e.g., if $Q(X, Y) = (Y - (X - 1))(Y^2 + X^3)$ then $R_X(Y) = a_3 Y^3 + a_2 \cdot Y^2 + a_1 \cdot Y + a_0$, where $a_3 = 1$, $a_2 = -X + 1$, $a_1 = X^3$ and $a_0 - X^4 + X^3$). Now note that $Y - f(X)$ is a factor of $Q(X, Y)$ if and only if $f(X)$ is a *root* of the univariate polynomial $R_X(Y)$, that is, the polynomial $R_Y(f(X))$ is the same as the zero polynomial (in the example, $Y - (X - 1)$ divides $Q(X, Y)$ and $R_X(X - 1) \equiv 0$).

Let us now return to problem of list decoding folded RS code with $m = 4$. Given the received word whose ith symbol (for $0 \le i < N$) is $(y_{i,0}, y_{i,1}, y_{i,2}, y_{i,3})$, we need to output all the close-by folded RS codewords. To motivate the idea behind the algorithm, for the time being assume that the transmitted codeword was from the so-called *interleaved* RS code of *order* 4. Any codeword in such a code will have as its ith symbol $(0 \le i \le N - 1)$ the 4-tuple $(f(\gamma^{4i}), f_1(\gamma^{4i}), f_2(\gamma^{4i}), f_3(\gamma^{4i}))$, where $f(X), f_1(X), f_2(X)$ and $f_3(X)$ are some polynomials of degree at most $k - 1$. We remark that the folded RS code is a subcode[e] of the interleaved RS code where $f_j(X) = f(\gamma^j X)$ for $1 \le j \le 3$.

Given the setup above, the first thing to explore is whether one can generalize the GS algorithm to the setting of interleaved RS codes. To see one such generalization, note that RS codes are interleaved RS codes of order 1. The GS algorithm interpolated a nonzero bivariate polynomial $Q(X, Y)$ in this case. Thus, for an interleaved RS code of order 4, a natural attempt would be to compute a nonzero 5-variate polynomial $Q(X, Y, Z, U, W)$, where (as before) Y is a placeholder for $f(X)$ and (this part is the generalization) Z, U, and W are placeholders for $f_1(X)$, $f_2(X)$, and $f_3(X)$, respectively. For the next step of root finding, we compute the 4-variate polynomial $R_X(Y, Z, U, W)$ that is equivalent to $Q(X, Y, Z, U, W)$. Now the hope would be to find out all the tuples (Y, Z, U, W) that make $R_X(Y, Z, U, W)$ vanish and that the required tuple $(f(X), f_1(X), f_2(X), f_3(X))$ is one of them. The latter condition can in fact be satisfied, but the trouble is that the number of tuples that make R_X zero could be very large (growing exponentially in n). To see intuitively what goes wrong, recall that in the Guruswami–Sudan setting, we had one unknown Y and one constraint $R_X(Y) = 0$. However, in the interleaved RS setting, we have *four* unknowns Y, Z, U, W but *only one* constraint $R_X(Y, Z, U, W) = 0$. This essentially means that three of the four unknowns are unconstrained and can thus be almost any polynomial of degree less than k.

The generalization above (and similar ideas) were tried out in a few works, but could not decode beyond the $1 - \sqrt{R}$ radius. Finally, 7 years after the GS algorithm was published, Parvaresh and Vardy[14] had an ingenious idea: force the polynomials $f_1(X), f_2(X)$ and $f_3(X)$ to be related to $f(X)$. In particular, they only look at the subcode of the interleaved RS code where $f_j(X) = (f_{j-1}(X))^d \bmod (E(X))$ for $1 \le j \le 3$ (we set $f_0(X) = f(X)$), for some positive integer parameter d and an irreducible polynomial $E(X)$. The reason we compute the modulus using an irreducible polynomial is that the relationships between these polynomials translate to the following relationships between their corresponding placeholders: $Z = Y^d$, $U = Y^{d^2}$, and $W = Y^{d^3}$. In other words, we gain *three new* constraints on the four variables Y, Z, U, W. Together with the interpolation constraint $R_X(Y, Z, U, W) = 0$, this restores equality in the number of unknowns and the number of constraints. This in turn implies that the number

[e] This is the same as looking at an appropriate subset of messages.

of possible solutions is polynomially bounded. (There are some steps involved to obtain this conclusion but they are mostly all "low-level details.") Further, this method not only establishes a bound on the number of solutions, but also gives a polynomial time algorithm to find these solutions. To see this, note that given the three new constraints, we are looking for roots of the *univariate* polynomial $R_X(Y, Y^d, Y^{d^2}, Y^{d^3})$, which can be accomplished by well-known polynomial time algorithms.[1]

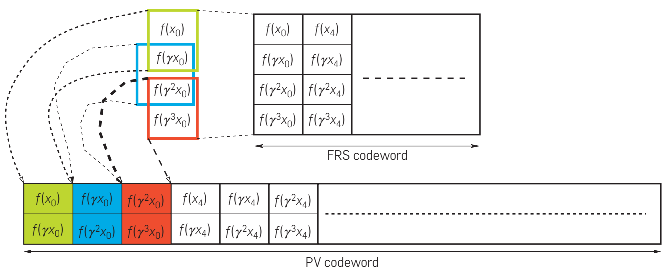
Finally, let us return to the problem of list decoding folded RS codes with $m = 4$. In folded RS codes also, we have the property that $f_j(X)$ is related to $f(X)$ for $1 \le j \le 3$. In fact, combining a couple of well-known results in finite fields, in Guruswami and Rudra[9] we show that $f(\gamma X) = (f(X))^{|\mathbb{F}|-1} \bmod (E'(X))$, where $E'(X) = X^{|\mathbb{F}|-1} - \gamma$ is an irreducible polynomial. We remark that the irreducible polynomial $E(X)$ in the Parvaresh–Vardy (henceforth, PV) codes only has some degree requirements. Our restriction on $E'(X)$ is stricter and thus, folded RS codes are a special case of PV codes. However, we are not done yet. Until now all we can claim is that folded RS code of rate R with folding parameter $m = 4$ can be list decoded from the same fraction of errors as the corresponding PV codes, which happens to be $1 - \sqrt[5]{(4R)^4}$. We have $4R$ appearing instead of R because the rate of the PV code is 1/4'th the rate of the original RS code, since the encoding of the message $f(X)$ now consists of the evaluations of *four* polynomials instead of just those of $f(X)$. Next we expand on our other main idea which "compresses" the PV encoding to avoid this rate loss, and enables correcting close to a fraction $1 - R$ of errors.

3.2. The final piece

The improvement over Parvaresh–Vardy comes from comparing apples to oranges. In particular, till now we have seen that the folded RS code with folding parameter 4 is a special case of the PV code of order 4. Instead let us compare the folded RS code with a PV code of *smaller* order, say 2. It turns out that the folded RS code with folding parameter 4 are compressed forms of certain specific PV codes of order 2 and we will exploit this observation. In particular, as in Figure 7 compare the folded RS codeword with the PV code of order 2 (where the polynomial $f(X)$ is evaluated at the points $\{1, \gamma, \ldots, \gamma^{n-1}\} \setminus \{\gamma^3, \gamma^7, \ldots, \gamma^{n-1}\}$). We find that in the PV encoding of f, for every $0 \le i \le n/m - 1$ and every $0 < j < m - 1$, $f(\gamma^{mi+j})$ appears exactly twice (once as $f(\gamma^{mi+j})$ and another time as $f_1(\gamma^{-1}\gamma^{mi+j})$), whereas it appears only once in the folded RS encoding. In other words, the information contained in one symbol in the folded RS codeword (which is worth *four* elements from $\mathbb{F}$) is repeated over three symbols in the PV codeword (which is worth *six* elements from $\mathbb{F}$). This implies that even though both the folded RS codeword and the PV codeword have exactly the same information, the folded RS codeword is compressed by a factor of 3/2. This in turn bumps up the rate of the folded RS code by the same factor. Hence, we can list decode folded RS codes with folding parameter 4 and rate R from a fraction $1 - \sqrt[3]{(4R/3)^2}$ of errors.

Thus, our list decoding algorithm for folded RS with folding parameter m can be modularly defined as follows:

Figure 7: The correspondence between a folded RS code (with $m = 4$ and $x_i = \gamma^i$) and the PV code (of order $s = 2$) evaluated over $\{1, \gamma, \gamma^2, \gamma^4, \ldots, \gamma^{n-4}, \ldots, \gamma^{n-2}\}$. The correspondence for the first block in the folded RS codeword and the first three blocks in the PV codeword is shown explicitly in the left corner of the figure.



unfold the received word for the appropriate PV code of order $s \le m$ and then run the Parvaresh–Vardy list decoder on this unfolded received word. It turns out that this list decoder can correct such a folded RS code of R from up to $1 - \sqrt[s+1]{\left(\frac{mR}{m-s+1}\right)^s}$ fraction of errors. By picking m to be (somewhat) larger than s and picking s to be sufficiently large (in terms of $1/\varepsilon$), we can conclude the following result.

THEOREM 1. *For every $\varepsilon > 0$ and $0 < R < 1$, there is a family of folded RS codes that have rate at least R and which can be list decoded up to a fraction $1 - R - \varepsilon$ of errors in time $(N/\varepsilon^2)^{O(\varepsilon^{-1}\log(1/R))}$ where N is the block length of the code. The alphabet size of the code is $(N/\varepsilon^2)^{O(1/\varepsilon^2)}$.*

We note that the time complexity has an undesirable dependence on ε, with $1/\varepsilon$ in the exponent. Improving this bound remains a challenging open question.

4. DECODING OVER SMALL ALPHABETS

So far we have discussed codes over large alphabets. For example, folded RS codes of rates R that can be list decoded from $1 - R - \varepsilon$ fraction of errors need alphabet size of roughly $n^{O(1/\varepsilon^2)}$, where n is the block length of the code. This large alphabet size can be a shortcoming. Next, we discuss known techniques that help us reduce the alphabet size.

We start with perhaps the most natural small alphabet: $\{0, 1\}$. For codes defined over this alphabet (also called *binary* codes), it turns out that to list decode from ρ fraction of errors the best possible rate is $1 - H(\rho)$, where $H(x) = -x\log_2 x - (1 - x)\log_2(1 - x)$ is the entropy function. Two remarks are in order. First, the rate $1 - H(\rho)$ is much smaller than the rate of $1 - \rho$ that folded RS codes can achieve. (It turns out that to attain a rate of $1 - \rho - \varepsilon$, the alphabet size needs to be at least $2^{1/\varepsilon}$; more on this later in the section.) Second, as shown in Shannon's seminal paper,[16] the quantity $1 - H(\rho)$ is *exactly* the same as the best possible rate (aka "capacity") that can be achieved in the binary symmetric channel BSC_ρ. Thus, list decoding can bridge the traditionally perceived gap between the Shannon stochastic model and the Hamming worst-case model.

We "transfer" our result for folded RS codes to a result for binary codes via a natural method for composing together codes called *code concatenation*, proposed by Forney over

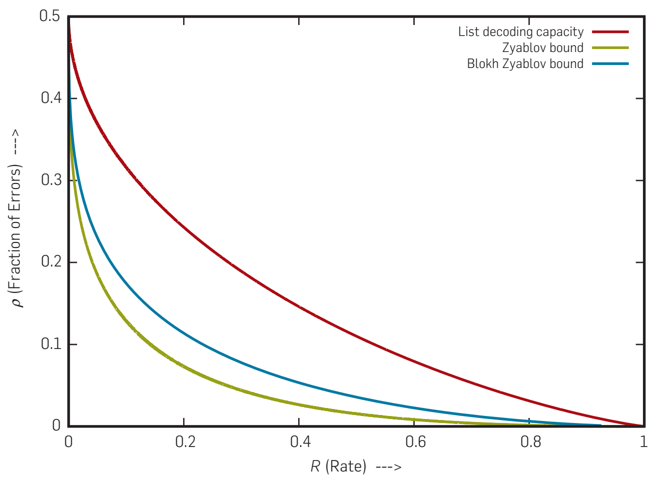
40 years ago. Thanks to the powerful algorithms for decoding folded RS codes, we can use this approach to achieve a certain trade-off called the *Zyablov bound* between rate and fraction of errors corrected.[9] In a subsequent work, using another generalization of code concatenation, we improved the trade-off to the *Blokh–Zyablov bound*.[8] Figure 8 plots these two trade-offs along with the best possible trade-off (list-decoding capacity). There is a large gap between the list-decoding capacity of binary codes and the best bound known to be achievable with efficient algorithms. Closing this gap remains a central and extremely challenging open question.

We now briefly mention how we resolve the large alphabet issue that was raised in Section 3. When the folding parameter of the folded RS code is a constant (as in Theorem 1), the number of bits needed to represent a symbol from the alphabet is no larger than roughly the logarithm of the block length of the folded RS code. This is small enough to use the idea of code concatenation mentioned above to reduce the alphabet. In order to maintain the optimal trade-off between rate and fraction of errors list decoded, we need to combine concatenation with an approach to redistribute symbols using *expander graphs*.[6] This leads to codes of rate R that can be list decoded from a fraction $1 - R - \varepsilon$ of errors over an alphabet of size $2^{1/\varepsilon^4}$, which is close to the lower bound of $2^{1/\varepsilon}$ mentioned earlier.

5. CONCLUDING REMARKS

First, we mention some related work that appeared subsequent to the initial publication of our result. Further work on extending the results in this article to the framework of *algebraic-geometric codes* has been done in Guruswami[5] and Guruswami and Patthak.[7] A surprising application of the ideas in the Parvaresh–Vardy list decoder is the construction of *randomness extractors* by Guruswami, Umans, and Vadhan.[11] Randomness extractors convert input from a weakly random source into an almost perfectly random string, and have been intensively studied in theoretical computer science for over 15 years. This recent extractor is almost optimal in all parameters, while having a simple, self-contained description and proof.

Figure 8: Error-correction radius ρ of our algorithms for binary codes plotted against the rate R. The best possible trade-off, i.e., list decoding capacity, is $\rho = H^{-1}(1 - R)$, and is also plotted.



Even though the work presented in this article makes good progress in our theoretical understanding of list decoding, applying these ideas into practice requires further innovation. We conclude by posing two practical challenges.

The first challenge is specific to making the list decoding algorithms for folded RS codes more practical. Recall that the algorithm involved an interpolation step and a "root-finding" step. There are fast heuristic approaches for the latter step that could be used in practice. The interpolation step, however, seems too inefficient for practical purposes due to the large size of the linear systems that need to be solved. It would be very useful to have more efficient algorithms for this step. We note that such improvements for the Guruswami–Sudan algorithm have been obtained.

The second challenge is more general. Codes have found numerous practical applications in domains such as communication and data storage. Despite its promise and the recent advances, list decoding has not yet found widespread use in practical systems (though as mentioned earlier, the Moonbounce program does use the multiplicities based list decoder). One possible reason could be that the previous list decoding algorithms do not provide much gain for the high rate regime over traditional unique decoding algorithms. However, this is no longer a concern—we now have algorithms that obtain much better theoretical bounds in this regime. Further, folded RS codes are very similar to RS codes that are ubiquitous in practice. Hopefully in the near future, list decoding will be used more widely in practical systems.

Acknowledgments

The research described here was supported in part by NSF award CCF-0343672 and fellowships from the Sloan and Packard Foundations. We thank Ronitt Rubinfeld for several valuable comments on an earlier draft of this paper.

References

1. Berlekamp, E. Factoring polynomials over large finite fields. *Math. Comput. 24* (1970), 713–735.
2. Elias, P. List decoding for noisy channels. *Technical Report 335*, MIT Research Lab of Electronics, 1957.
3. Gemmell, P., Sudan, M. Highly resilient correctors for multivariate polynomials. *Information Processing Lett. 43*, 4 (1992), 169–174.
4. Goldreich, O., Levin, L. A hard-core predicate for all one-way functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, 1989, 25–32.
5. Guruswami, V. Artin automorphisms, cyclotomic function fields, and folded list-decodable codes, 2008. Manuscript.
6. Guruswami, V., Indyk, P. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. Information Theory 51*, 10 (2005), 3393–3400.
7. Guruswami, V., Patthak, A. Correlated algebraic-geometric codes: Improved list decoding over bounded alphabets. *Math. Comput. 77*, 261 (Jan. 2008), 447–473.
8. Guruswami, V., Rudra, A. Better binary list-decodable codes via multilevel concatenation. In *Proceedings of the 11th International Workshop on Randomization and Computation*, 2007, 554–568.
9. Guruswami, V., Rudra, A. Explicit codes achieving list decoding capacity: Error-correction up to the Singleton bound. *IEEE Trans. Information Theory 54*, 1 (2008), 135–150. Preliminary version in *STOC'06*.
10. Guruswami, V., Sudan, M. Improved decoding of Reed–Solomon and algebraic-geometric codes. *IEEE Trans. Information Theory, 45* (1999), 1757–1767.
11. Guruswami, V., Umans, C.,

Vadhan, S.P. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. In *Proceedings of 22nd IEEE Conference on Computational Complexity*, 2007, 96–108.
12. Hamming, R.W. Error detecting and error correcting codes. *Bell System Technical J. 29* (Apr. 1950), 147–160.
13. Koetter, R., Vardy, A. Algebraic soft-decision decoding of Reed–Solomon codes. *IEEE Trans. Information Theory 49*, 11 (2003), 2809–2825.
14. Parvaresh, F., Vardy, A. Correcting errors beyond the Guruswami–Sudan radius in polynomial time. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, 2005, 285–294.
15. Peterson, W.W. Encoding and error-correction procedures for Bose–Chaudhuri codes. *IEEE Trans. Information Theory*, 6 (1960), 459–470.
16. Shannon, C.E. A mathematical theory of communication. *Bell System Technical J. 27* (1948), 379–423, 623–656.
17. Sudan, M. Decoding of Reed–Solomon codes beyond the error-correction bound. *J. Complexity*, 13, 1 (1997), 180–193.
18. Welch, L.R., Berlekamp, E.R. Error correction of algebraic block codes. *US Patent Number 4,633,470*, December 1986.
19. Wozencraft, J.M. List Decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48 (1958), 90–95.

Venkatesan Guruswami (venkat@cs.washington.edu), Computer Science and Engineering, University of Washington, Seattle, WA 98105, USA Currently visiting the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Atri Rudra (atri@cse.buffalo.edu), Computer Science and Engineering, University at Buffalo, SUNY, Buffalo, NY.

DOI:10.1145/1467247.1467270

Technical Perspective Where Biology Meets Computing

By Bud Mishra

ALAN TURING DIED in 1954 in his laboratory after eating a cyanide-laced apple. Though Turing's mother believed her son's death to be a result of the kind of accidents that befalls absent-minded mathematicians engaged in laboratory experiments, it is generally assumed to be a suicide.

During his last years, Turing had become an experimentalist, interested in bio-chemical systems. He had proposed a reaction-diffusion model in his 1952 paper entitled "The Chemical Basis of Morphogenesis," putting forth his hypothesis of biological pattern formation. Turing's models describe how the concentration of certain substances (called morphogens) distributed in space change under two continuous-time processes: *local chemical* reactions, in which the substances are converted into each other, and *diffusion*, which causes the substances to spread out in space. The solutions to Turing's Reaction-Diffusion equation display diverse patterns such as traveling waves, spirals, spots, stripes and dissipative solitons. Turing's models focused on only continuously varying concentrations of morphogens: he famously wrote, "since the role of genes is presumably catalytic, ...they may be eliminated from the discussion."

However, genes turned out to be far more important in biological pattern formation. Triggered by a small group of transcriptional activators (proteins and microRNAs), the genes turn themselves on and off in a complex but tightly programmed choreography and control the concentration and spatial distribution of many biomolecules, including the transcriptional activators. Thus, pattern formation in biology is better understood by hybrid automata, in which the genes form complex discrete modes with their own program for state-transitions, while exhibiting continuous dynamics as the system dwells in various modes.

Another interesting characteristics of pattern formation is captured nicely in Wolpert's French-Flag (or PI, Positional Information) Model, where the discrete levels of morphogen-concentration gradients, varying complexly over space and time, determine the fates of the biological cells in the local neighborhoods. This model is highly robust, scale-invariant, and asynchronous; they exhibit temporal structures in which order of the events are far more important than their exact timing. Thus, while the genotype/syntax of these systems are easily described by hybrid automata, their phenotype/semantics can be ideally described by temporal logics.

As luck would have it, a growing community of computer scientists has been thinking about problems like these for last few decades and developing many powerful model-checking tools to debug complex asynchronous systems. Many of these researchers have now turned to systems biology, as exemplified by the paper here by Grosu et al.

The authors describe a biological model of interacting heart cells and studies how they form complex electrical patterns, using model-checking and machine-learning tools for specification, learning, and detection of emergent behavior/patterns in networks of hybrid automata. These tools shed important light on the process of atrial fibrillation (Afib), an abnormal rhythm originating in the upper chambers of the heart and afflicting millions, with incidences increasing with age. The cardiac tissue is a spatial network of myocytes (muscle fiber cells) that must contract in a coordinated fashion in order to pump blood effectively. Coordination is ensured through a reaction-diffusion system (RDS): the pace-making myocytes generate an electric stimulus that diffuses to the neighboring myocytes; these react in an all-or-nothing fashion, which reinforces the stimulus and ensures its further propagation without damping. Reaction is governed by specific molecules (ion channels) in the myocyte membrane. The authors introduce many innovations to attack this problem algorithmically, namely, they replace the standard Luo-Rudi model of nonlinear partial differential equations by a network of hybrid automata and analyze them through efficient mode-abstractions and superposition; they develop a new modal logic, based on spatial-superposition, for specifying emergent behavior; they devise an ingenious method for learning the formulae of this logic from the spatial patterns; and finally, apply bounded model checking to detect the onset of one such biomedically important emergent patterns, that is, spiral waves.

The authors lead one to believe that the future of computer science very likely lies not just in devising powerful tools to catalyze large-scale experiments or to warehouse massive amount of experimental data to be searched and mined, but also as an interpreter and re-describer of complex phenomena. In this role, using tools described here, computer scientists can revolutionize the way we attempt to understand a large tangle of interconnected neurons, a large social-network of presumably altruistic individuals, a crowd responding to a catastrophe, a global financial market interacting through complex trades, an interconnected power-grid, and so on. We could try to understand their topology, structural evolution, spatial patterning, self-organization, stochasticities, causal links, and emergent behaviors. We could look for design principles in these complex systems, some of which are thought (by some) to have been crafted by an intelligent designer, who appears to have cavalierly released these systems without proper documentation. C

Bud Mishra (mishra@nyu.edu) is a professor of computer science, mathematics, and cell biology at New York University's Courant Institute and School of Medicine. He is also a visiting scholar at the Cold Spring Harbor Laboratory and a fellow of both ACM and IEEE.

DOI:10.1145/1467247.1467271

Learning and Detecting Emergent Behavior in Networks of Cardiac Myocytes

By Radu Grosu, Scott A. Smolka, Flavio Corradini, Anita Wasilewska, Emilia Entcheva, and Ezio Bartocci

Abstract

We address the problem of specifying and detecting emergent behavior in networks of cardiac myocytes, spiral electric waves in particular, a precursor to atrial and ventricular fibrillation. To solve this problem we: (1) apply discrete mode abstraction to the cycle-linear hybrid automata (CLHA) we have recently developed for modeling the behavior of myocyte networks; (2) introduce the new concept of spatial superposition of CLHA modes; (3) develop a new spatial logic, based on spatial superposition, for specifying emergent behavior; (4) devise a new method for learning the formulae of this logic from the spatial patterns under investigation; and (5) apply bounded model checking to detect the onset of spiral waves. We have implemented our methodology as the EMERALD tool suite, a component of our EHA framework for specification, simulation, analysis, and control of excitable hybrid automata. We illustrate the effectiveness of our approach by applying EMERALD to the scalar electrical fields produced by our CELLEXCITE simulation environment for excitable-cell networks.

1. INTRODUCTION

One of the most important and intriguing questions in systems biology is how to formally specify *emergent behavior in biological tissue*, and how to efficiently predict and detect its onset. A prominent example of such behavior is electrical *spiral waves* in spatial networks of cardiac myocytes (heart cells). Electrical impulses regularly circulate through cardiac tissue and cause the heart's muscle fibers to contract. In a healthy heart, these electrical impulses travel smoothly and unobstructed, like a water wave that ripples gently in a pond. These waves can, however, sometimes develop into troublesome, whirlpool-like spirals of electrical activity. Spiral waves of this nature are a precursor to a variety of cardiac disturbances, including *atrial fibrillation* (AF), an abnormal rhythm originating in the upper chambers of the heart. AF afflicts two–three million Americans alone, putting them at risk for clots and strokes. Moreover, the likelihood of developing AF increases with age.

In this paper, we address this question by proposing a simple and efficient method for learning, and automatically detecting the onset of, spiral waves in cardiac tissue. See Figure 1 for an overview of our approach. Underlying our method is a *linear spatial-superposition logic* (LSSL) we have developed for specifying properties of spatial networks. LSSL is discussed in greater detail below. Our method also builds upon hybrid automata, image processing, machine learning, and model-checking techniques to first learn an LSSL formula that characterizes such spirals. The formula is then automatically checked against a *quadtree* representation[20] of the scalar electrical field (SEF) produced at each discrete time step by a simulation of a hybrid-automata network modeling the myocytes. A scalar field is a function that associates a scalar value, which in our case is an electric potential, to every point in space. The quadtree representation is obtained via discrete mode abstraction and hierarchical superposition of the elementary units within the SEF.

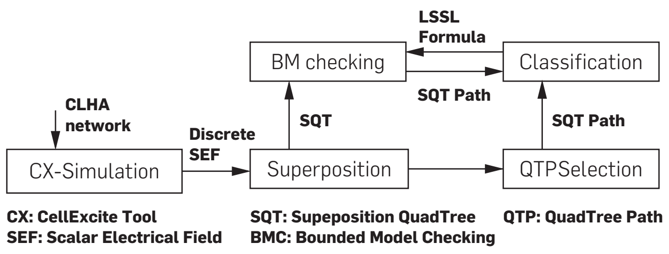
The electric behavior of cardiac myocytes is hybrid in nature: they exhibit an all-or-nothing electrical response, the so-called *action potential* (AP), to an external excitation. An AP can thus be viewed as triggering a discrete mode transition from the cell's resting mode of continuous behavior to its excited mode of continuous behavior. Despite their discrete-continuous hybrid nature, networks of myocytes have traditionally been modeled using nonlinear partial differential equations.[13, 17] While highly accurate in describing the molecular processes underlying cell behavior—nonlinear differential equations allow one to closely match the values of a multitude of state variables to their actual physical values—these models are not particularly amenable to formal analysis and typically do not scale well for the simulation of complex cell networks.

In Grosu et al.,[11] we showed that it is possible to automatically learn a much simpler *hybrid automaton* (HA)[12] model for cardiac myocytes, which explicitly captures, up to a prescribed error margin, the mixed discrete and continuous behavior of the AP. To highlight its *cyclic* structure and its *linear* dynamics, which may vary in interesting ways from cycle to cycle, we called it a *cycle-linear hybrid automaton* (CLHA). Moreover, one can use a variant of this CLHA model to efficiently (up to an order of magnitude faster) and accurately simulate the behavior of myocyte networks, and, in particular, induce spirals and fibrillation.[2,24,25]

A key observation concerning our simulations, see Figure 3, is that mode abstraction, in which the AP value of each CLHA in the network is *abstracted* to its corresponding mode, faithfully preserves the network's waveform and other spatial characteristics. Hence, for the purpose of learning, and detecting the onset of, spirals within CLHA networks, we can exploit mode abstraction to dramatically reduce the system state space. A

An earlier version of this paper appeared in *Proc. 11th International Conference on Hybrid Systems: Caomputation and Control* (HSCC'08), Springer, *LNCS* 4981, April 2008.

Figure 1: Overview of our method.



similar mode abstraction is possible for voltage recordings in live cell networks.

The state space of a 400×400 CLHA network is still prohibitively large, even after applying the above-described abstraction: it contains $4^{160,000}$ modes, as each CLHA has four mode values. To combat state explosion, we use a spatial abstraction inspired by Kwon and Agha[14]: we regard the mode of each automaton as a probability distribution and define the *mode superposition* of a set of CLHAs as the probability that an arbitrary CLHA in this set is in a particular mode. By successively applying superposition to the network, we obtain a tree structure, the root of which is the mode superposition of the entire CLHA network, and the leaves of which are the modes of the individual CLHA. The particular structure we employ, quadtrees, is inspired by image-processing techniques.[20] We shall refer to quadtrees obtained in this manner as *superposition-quadtrees* (SQTs).

LSSL is an appropriate logic for reasoning about *paths* in SQTs, and the spatial properties of CLHA networks in which we are interested, including spirals, can be cast in LSSL. For example, we have observed that the presence of a spiral can be formulated in LSSL as follows: Given an SQT, is there a path from its root leading to the core of a spiral? Based on this observation, we build a machine-learning classifier, the training-set records for which correspond to the probability distributions associated with the nodes along such paths. Each distribution, for mode value *stimulated*, corresponds to an attribute of a training-set record, with the number of attributes bounded by the depth of the SQT. An additional attribute is used to classify the record as either spiral or nonspiral. For spiral-free SQTs, we simply record the path of maximum distribution.

For training purposes, our CELL EXCITE simulator[2] generates, upon successive time steps, snapshots of a 400×400 CLHA network and their mode abstraction; see Figures 1 and 3. Training data for the classifier is then generated by converting the abstracted snapshots into SQTs and selecting paths leading to the core of a spiral (if present). The resulting table is input to the decision-tree algorithm of the Weka machine-learning tool suite.[8] This produces a classifier, in the form of a path-predicate, constraining the distribution of the attribute *stimulated* in each node along the path.

The syntax of LSSL is similar to that of linear temporal logic, with LSSL's Next operator corresponding to *concretization* (anti-superposition). Moreover, a sequence of LSSL Next operators corresponds to a path through an SQT. The classifier produced by Weka can therefore be regarded as an LSSL formula. An SQT path can be thought of as a magnifying glass, which starting from the root produces an increasingly detailed but more focused view of the image (i.e., abstracted snapshot). This effect is analogous to *concept hierarchy* in data mining[16] and arguably similar to the way the brain organizes knowledge: a human can recognize a word or a picture without having to look at all of the characters in the word or all of the details in the picture, respectively.

Although the LSSL logic and its underlying semantics (Kripke structures) allow us to reason about infinite paths through recursive structures (fractals), physical considerations—such as the number of myocytes in a cardiac tissue or the screen resolution—impose a maximum length k on such paths. We therefore maintain k as a parameter in LSSL's semantic definition, permitting us to accommodate any finite number of myocytes or screen resolution. Defining LSSL's semantics in this manner places us within the framework of bounded model checking.[3]

Our spatial-superposition logic might also be understood as a *Scale logic*, as it allows us to examine an image at various scales or levels of detail. The notion of scale is prevalent in biological systems, ranging from genetic scale to societal scale. The built-in notion of scale in our logic therefore makes it well suited for reasoning about biological systems.

We are now in a position to view spiral detection as a bounded-model-checking problem[3]: Given the SQT Q generated from the discrete SEF of a CLHA network and an LSSL formula φ learned through classification, is there a finite path π in Q satisfying φ? We use this observation to check every time step during simulation whether or not a spiral has been created. More precisely, the LSSL formula we use states that no spiral is present, and we thus obtain as a counterexample one or all the paths leading to the core of a spiral. In the latter case, we can identify the number of spirals in the SEF and their actual position.

The above-described method, including user-guided path selection, has been fully implemented as the EMERALD tool suite for automated spiral learning and detection. EMERALD is written in Java, and it is a new component of our EHA environment for the specification, simulation, analysis, and control of CLHA networks. EHA stands for Excitable Hybrid Automata, as we have used CLHA to model various types of excitable cells, including neurons and cardiac myocytes.[25] The EHA environment is freely available from Grosu et al.[10]

The rest of the paper is organized as follows. Section 2 reviews excitable-cell networks and their modeling with CLHA. Section 3 defines superposition and quadtrees, the essential ideas underlying LSSL, the topic of Section 4. Section 5 describes our learning and bounded-model-checking techniques; their implementation is considered in Section 6, along with our experimental results. Section 7 discusses related work. Section 8 offers our concluding remarks and directions for future research.

2. BIOLOGICAL BACKGROUND

An *excitable cell* (EC) has the ability to propagate an electrical signal, known at the cellular level as the *action potential* (AP), to neighboring cells. An AP corresponds to a change of potential across the cell membrane, and is caused by the flow of ions between the inside and outside of the cell through the membrane's ion channels.

An AP is an externally triggered event (with duration): an EC fires an AP as an all-or-nothing response to a suprathreshold stimulus, and each AP follows the same sequence of phases (described below) and exhibits roughly the same waveform regardless of the applied stimulus. During most of the AP no re-excitation can generally occur (the EC is in a refractory period).

Despite differences in duration, morphology, and underlying ion currents, the following major AP *phases* can be identified across different species and EC types: *resting*, *stimulated*, *upstroke*, *early repolarization*, *plateau*, and *final repolarization*. We abbreviate them as `r` (resting and final repolarization), `s` (stimulated), `u` (upstroke), and `p` (plateau and early repolarization).

Using the AP phases as a guide, we have developed, for several representative EC types, CLHA models that approximate the AP and other bioelectrical properties with reasonable accuracy. Their derivation was first performed manually.[24, 25] We subsequently showed in Grosu et al.[11] how to fully automate this process by learning various biological aspects of the AP of different cell types.

Intuitively, a hybrid automaton[12] is an extended finitestate automaton, the states of which encode the various phases of continuous dynamics a system may undergo, and the transitions of which are used to express the switching logic between these dynamics. The CLHA we obtained are fairly compact in nature, employing two or three continuous state variables and four to six modes. The term *cycle-linear* is used to highlight the cyclic structure of CLHA, and the fact that while in each cycle they exhibit linear dynamics, the coefficients of the corresponding linear equations and mode-transition guards may vary in interesting ways from cycle to cycle.

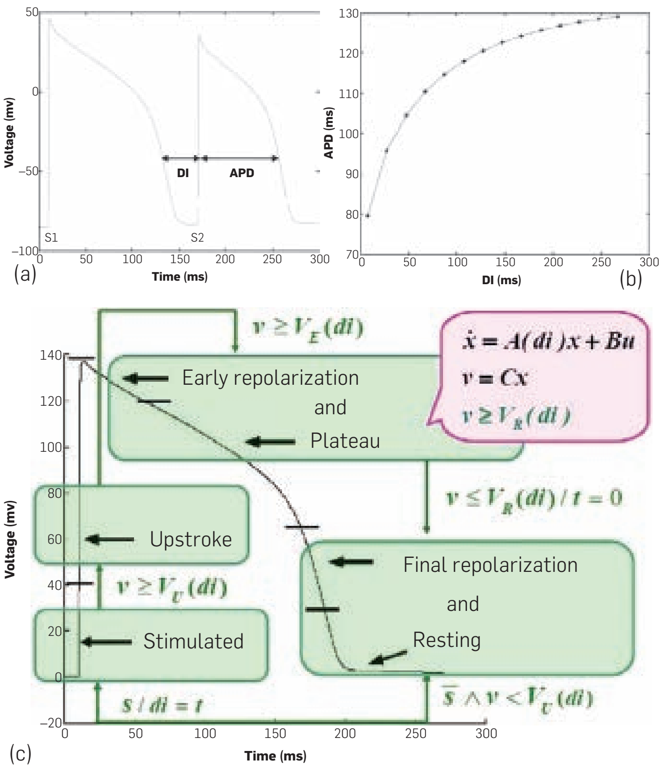
Figure 2 presents one of our CLHA models. To understand the model, first note that when an EC is subjected to repeated stimuli, two important time periods can be identified: AP *duration* (APD), the time the cell is in an excited state, and *diastolic interval* (DI), the time between the "end" of the AP and the next stimulus. Figure 2(a) illustrates the two intervals. The function relating APD to DI is called the APD *restitution function*. As shown in Figure 2(b), the relationship is nonlinear and captures the phenomenon that a longer recovery time is followed by a longer APD. A physiological explanation of a cell's restitution is rooted in the ion-channel kinetics as a limiting factor in the cell's frequency response.

The CLHA model itself, superimposed over the image of a typical AP, is given in Figure 2(c). Each mode has an associated linear dynamics $\dot{x} = Ax + Bu$, $v = Cx$, where x is the CLHA state, u is the input, and v (for voltage) is the output. A mode also has an associated invariant in v, forcing the outgoing transition to be eventually taken. The concept of mode dynamics and invariant is illustrated in Figure 2(c) for mode p (plateau and early repolarization); see that mode's callout. Transition labels are of the form $e \wedge g/a$, where e is an (optional) event, g is a guard, and $/a$ is an optional set of assignments. The only events in the model, representing the start and end of stimulation, are denoted by s and $\bar{s}$, respectively. Observe the per-mode and transition-guard dependence on the DI, which is measured with the help of clock variable t.

The dynamics of excitable-cell networks play an important role in the physiology of many biological processes. For cardiac cells, on each heart beat, an electrical control signal is generated by the sinoatrial node, the heart's internal pacemaking region. Electrical waves then travel along a prescribed path, exciting cells in atria and ventricles and assuring synchronous contractions. Of special interest are cardiac arrhythmias: disruptions of the normal excitation process due to faulty processes at the cellular level, single ion-channel level, or at the level of cell-to-cell communication. The clinical manifestation is a rhythm with altered frequency, tachycardia (rapid heart beat) or bradycardia (slow heart beat), or the appearance of multiple frequencies, polymorphic ventricular tachycardia, with subsequent deterioration to a chaotic signal, ventricular fibrillation (VF). VF is a typically fatal condition in which there is uncoordinated contraction of the cardiac muscle of the ventricles in the heart. As a result, the heart fails to adequately pump blood and hypoxia (lack of oxygen) may occur.

In order to simulate the emergent behavior of cardiac tissue, we have developed CELLEXCITE,[2] a CLHA-based simulation environment for excitable-cell networks. CELLEXCITE allows the user to sketch a tissue of excitable cells, plan the stimuli to be applied during simulation, and customize the arrangement of cells by selecting the appropriate lattice. Figure 3 presents our simulation results for a 400×400 CLHA network of cardiac myocytes. Nine 50-ms simulation steps are shown, during which (steps 1 and 4) the network was stimulated twice, at different regions. The results we obtain demonstrate the feasibility of using CLHA networks to capture and mimic different spatiotemporal behavior of wave propagation

Figure 2: AP duration, restitution function, and CLHA model of cardiac myocytes.



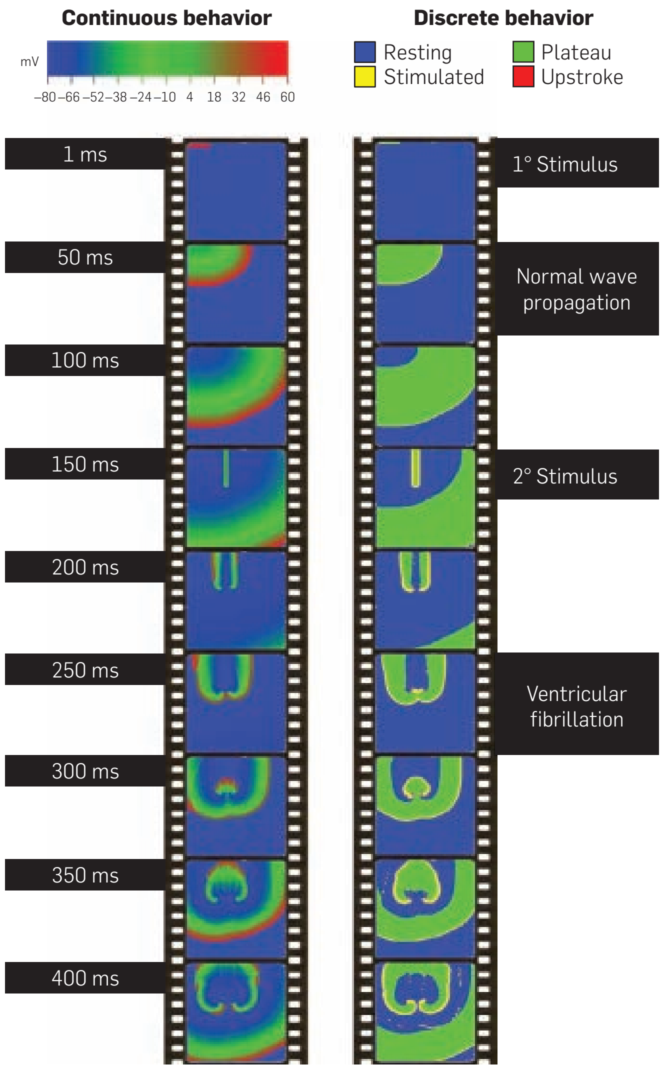
in 2D isotropic (homogeneous) cardiac tissue, including normal wave propagation (1–150 ms); the creation of spirals, a precursor to fibrillation (200–250 ms); and the break-up of such spirals into more complex spatiotemporal patterns, signaling the transition to VF (250–400 ms).

As can be clearly seen in Figure 3, mode abstraction, in which the action-potential value of each CLHA in the network is *discretely abstracted* to its corresponding mode, faithfully preserves the network's waveform and other spatial characteristics. Hence, for the purpose of learning and detecting spirals within CLHA networks, we can exploit mode abstraction to dramatically reduce the system state space.

3. SUPERPOSITION AND QUADTREES

A key benefit of using hybrid automata compared to nonlinear ODEs is their explicit support for finitary abstractions: the infinite range of values of a hybrid automaton's continuous state variables can be abstracted to the automaton's discrete finite set of modes. As discussed in Sections 1 and 2, abstracting the AP (voltage) of the constituent CLHA in a CLHA network to their corresponding mode (s, u, p, or r) turns out to faithfully preserve the network's waveform and other spatial characteristics. This simplifying approximation allows us to reduce the spiral-onset verification problem to a finite-state verification problem.

Figure 3: Simulation of continuous and discrete behavior of a CLHA network.



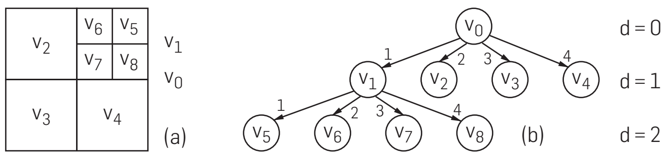
Although in this paper we consider a CLHA network an execution at a time, our ultimate goal is exhaustive simulation, i.e., model checking. Within this context, the state space of a 400×400 CLHA network, which would be necessary to simulate the behavior of a tissue of about 16 cm^2 in size, is still too large for analysis purposes: it has $4^{160,000}$ mode values! To combat state explosion, we use a spatial abstraction inspired by Kwon and Agha.[14] Consider the state space of a CLHA network. We regard the current mode of each CLHA in the network as a degenerate probability distribution, and define the *superposition* of a set of (possibly superposed) modes as the mean of their distributions. By successively applying superposition to the CLHA network, we obtain a tree whose root is the mode superposition of the entire network, and whose leaves are the individual modes of the component CLHA. The particular superposition tree structure we employ, the quadtree, was inspired by image-processing techniques.[20]

Let $\mathcal{A}$ be a $2^k \times 2^k$ matrix of CLHA modes. A quadtree $Q = (V, R)$ representation of $\mathcal{A}$ is a quaternary tree, such that each vertex $v \in V$ represents a sub-matrix of $\mathcal{A}$ and the transition relation R defines v's 4 child vertices (assuming v is not a leaf). For example, the root v_0 of the quadtree in Figure 4 represents the entire matrix; child v_1 represents the matrix $\{2^{k-1}, \ldots, 2^k\} \times \{0, \ldots, 2^{k-1}\}$; child v_6 represents the matrix $\{2^{k-1}, \ldots, 3 * 2^{k-2}\} \times \{0, \ldots, 2^{k-2}\}$; etc.

Due to superposition, a quadtree is in general a more efficient data structure than the matrix it represents: if the subtree rooted at a node of a quadtree is of one "color" (mode in our case), then there is no need to descend into the node's subtree as no additional information can be gleaned by doing so. Moreover, given a quadtree representation of an image and a property of the image in which one is interested—such as determining whether a mode-abstracted snapshot of a CLHA network contains a spiral—it may only be necessary to follow a *path* through the quadtree (as opposed to exploring the entire tree) to determine if the property holds. Moreover, the path need not necessarily descend all the way to the leaf level, but rather may terminate at an interior node. See Sections 4 and 5 for a further discussion of such quadtree properties.

Definition 1 (**Distributions**). Let $\mathcal{N}$ be a CLHA network whose constituent CLHA have (ordered) modes $M = \{$s, u, p, r$\}$, and let Q be the quadtree representation of $\mathcal{N}$. Then each leaf node $l \in Q$ has an associated degenerate *leaf distribution* D_l whose probability mass function (also sometimes known as the point mass function, and in either case abbreviated

Figure 4: Quadtree representation.



as PMF) p_l is such that $\exists! m \in M \,.\, p_l(m) = 1$. Also, let $i \in Q$ be an interior node with children $i_1, \ldots, i_4$. Then i has an associated *superposition distribution* D_i whose PMF p_i is such that $\forall m \in M \,.\, p_i(m) = \frac{1}{4} \sum_{j=1}^{4} p_{ij}(m)$.

The intuition is as follows. If a leaf node occurs at the maximum depth of the quadtree, then it corresponds to the current mode of a CLHA. As CLHA are deterministic, they assume one of the values in M with probability 1. (We will weaken this restriction at the end of the section when we consider superposition quad-graphs.) If the leaf does not occur at the maximum depth of the quadtree, then it corresponds to the superposition of identical degenerate distributions, and no additional information is obtained by decomposing the leaf into its four superposition components. The visual interpretation is that a pixel has one definite color, and nothing is learned by decomposing an area in which all pixels have the same color.

As for the distribution of an interior node i, if all of i's children are leaves, then, for each mode value m, i's superposition is the mean of the occurrences of m. Hence, the probability that the mode of the parent is m is the probability that the mode of an arbitrary child is m. If i's children are interior nodes, it still holds that the probability that i's mode is m is the probability that the mode of an arbitrary leaf below i's children is m.

We call a quadtree whose nodes are labeled with leaf- and interior-node distributions a *superposition quadtree* (SQT). The distributions in an SQT are not known in advance; our learning algorithm seeks to determine them for what we perceive to be spirals. The use of probability distributions is justified by the fact that different spirals might have slightly different shapes, i.e., slightly different distributions of values for the leaf nodes of their associated quadtrees.

The SQTs presented so far were constructed over a finite matrix A containing $2^k * 2^k$ elements. In general, SQTs can be obtained via the finite unfolding of a *quad-graph*.

Definition 2 (SQG). A *superposition quad-graph* (SQG) is a 4-tuple $G = (V, v_0, R, L)$ consisting of:

- a finite set of *vertices* V with *initial vertex* $v_0 \in V$,
- a *transition relation* $R \subseteq V \times [1..4] \times V$ s.t. $\forall v \in V, i \in [1..4].\, \exists\, u \in V.\, (v, i, u) \in R$, and
- a *probability-distribution labeling* L s.t. $\forall v \in V.\, L(v) = \frac{1}{4} \sum_{u \in R(v)} L(u)$.

The condition on R ensures that each vertex in V has precisely four successors in R. The condition on L ensures that the probability distributions are related through superposition. The manner in which we construct SQTs as finite unfoldings of SQGs can be extended to support the definition of *infinite* SQTs generated by *recursion*. That is, it supports the definition of *fractals*. Furthermore, just as we use SQTs to represent finite images, SQGs can be used to represent *infinite* images, i.e., fractals.

Figure 5(a–c) gives SQGs representing the recursive specification of three fractals and a graphical depiction of the unfolding of these SQGs up to depth 3. (The SQG of Figure 5(d), for which no depiction is given, is considered below.) Note the fractal-like nature of these pictures: the *gray* areas represent recursion and correspond to recursive nodes in the SQGs. Such nodes are labeled by *distribution variables*, the values for which can be computed by solving a *linear system*. For example, x and y in Figure 5(b) are computed by solving the linear system $x = 1/4\ (x + 1 + y)$ and $y = 1/4\ (1 + x)$. The four self-loops of the leaves are not shown for simplicity. Note that leaves may now be associated with any constant distribution. Also note that the finite-state SQGs of Figure 5 (b) and (d) yield equivalent infinite SQTs.

4. LINEAR SUPERPOSITION LOGIC

In this section, we define LSSL. Although the LSSL logic—especially its spatial analogues of the temporal fixpoint operators of LTL[18]—and its underlying semantics (Kripke structures) allow us to reason about infinite paths, physical considerations such as the number of myocytes in a cardiac tissue or screen resolution impose a maximum length k on paths. We therefore maintain k as a parameter in LSSL's semantic definition, placing us within the framework of bounded model checking.[3]

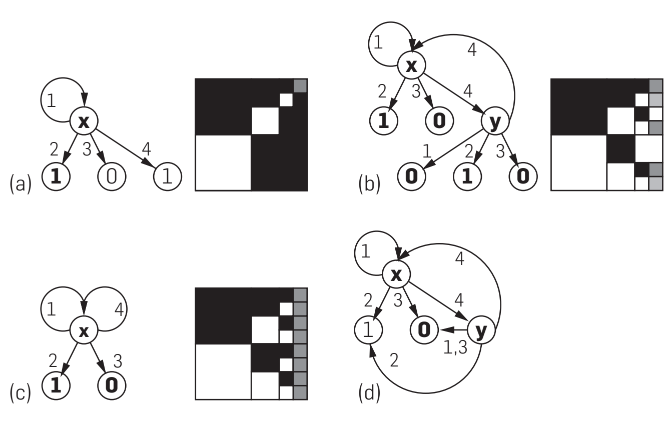
Every finite SQT can be transformed into an SQG by adding to each leaf node a self-loop labeled by i, $i \in [1..4]$. Moreover, an SQG can be transformed into a *Kripke structure* by erasing (forgetting) the transition labeling, collapsing all resulting transitions that share the same source and target nodes into one transition, and assuming nondeterminism among transitions emanating from the same node. For example, applying this forgetful transformation to the SQGs of Figure 5 yields the Kripke structures of Figure 6, where the self-loops are made explicit. The Kripke structure of Figure 6(d) can be seen as the minimal-state equivalent of the one of Figure 6(b) where nodes labeled by 0 or 1 are shared.

Definition 3 (**Kripke structure**). A *Kripke structure* (KS) over a set of atomic formulas AF is a four-tuple $M = (S, I, R, L)$ consisting of:

- a countable set of *states* S, with *initial states* $I \subseteq S$,
- a *transition relation* $R \in S \times S$ with $\forall s \in S.\, \exists t \in S.\, (s,t) \in R$, and
- a *labeling (or interpretation) function* $L : S \to 2^{AF}$.

The condition associated with the transition relation R ensures that every state has a successor in R. Consequently, it is always possible to construct an infinite path through a KS,

Figure 5: Fractals as finite-state SQGs: (a) $x = 2/3$, (b) $x = 5/11$, $y = 4/11$, and (c) $x = 1/2$. SQG (d) is equivalent to SQG (b).



an important property when dealing with reactive systems. In our case, it means that we can reason about recursive SQTs, i.e., fractals.

The labeling function L defines for each state $s \in S$ the set $L(s)$ of atomic formulas that are valid in s. Atomic formulas are *inequalities over distributions* of the form $P[D = m] \sim d$, where D is a distribution function, $m \in M$ is a discrete value (e.g. a mode), $d \in [0..1]$, and $\sim$ is one of $<, \leq, =, \geq$, or $>$. We use $P[D = m]$ as a more intuitive notation for $p(m)$, where p is the PMF associated with D. (This notation is also reminiscent of $P[X = m]$, where X is a random variable.) It should thus be noted that the 0–1 state labels used in Figure 6, where the mode in question is s, are shorthand for the atomic formula $P[D = \mathrm{s}] = 0$ or $P[D = \mathrm{s}] = 1$.

In order to verify properties of a reactive system modeled as a KS K, it is customary to use either a linear-time or a branching-time temporal logic. A model for a linear-time (LTL) formula is an infinite path π in K. A model for a branching-time formula is K itself; given a state s of K, this allows one to quantify over the paths originating from s. For our current purposes of specifying and detecting the onset of spirals, LTL suffices.

Strictly speaking, our logic is a *linear spatial-superposition logic* (LSSL), as a path π in K represents a sequence of *concretizations* (anti-superpositions). Syntactically, however, our temporal-logic operators are the same as in LTL: the *next* operator X, with $X\varphi$ meaning that φ holds in a concretization of the current state; its inverse operator B; the *until* operator U, with $\varphi U \psi$ meaning that φ holds along a path until ψ holds; and the *release* operator R, with $\psi R \varphi$ meaning that φ holds along a path unless released by ψ.

Definition 4 (LSSL **Syntax**). The syntax of *linear spatial-superposition logic* is defined inductively as follows:

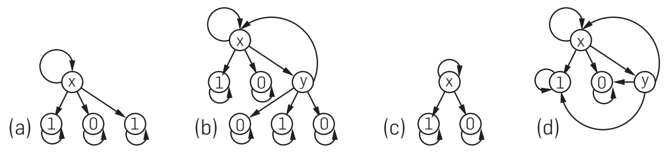
$$\begin{aligned}\varphi \;\; &::= \;\; \top \,|\, \bot \,|\, P[D=m] \sim d \,|\, \neg\phi \,|\, \varphi \vee \psi \,|\, X\varphi \,| \\ &\qquad B\varphi \,|\, \varphi U \varphi \,|\, \varphi R \varphi \\ \sim \;\; &::= \;\; < \,|\, \leq \,|\, = \,|\, \geq \,|\, >\end{aligned}$$

As discussed above, a bound k on the path length is maintained as a parameter in LSSL's semantic definition.

Definition 5 (LSSL **Semantics**). Let K be a KS, π a path in K, and $f \in AF$ an atomic formula. Then, for $k \geq 0$, *π satisfies an* LSSL *formula φ with bound k*, written $\pi \models_k \varphi$, only if $\pi \models_k^0 \varphi$, where:

$$\begin{array}{lcl}
\pi \models_k^i \top & \text{and} & \pi \not\models_k^i \bot \\
\pi \models_k^i f & \Leftrightarrow & f \in L(\pi[i]) \\
\pi \models_k^i \neg\varphi & \Leftrightarrow & \pi \not\models_k^i \varphi \\
\pi \models_k^i \varphi \vee \psi & \Leftrightarrow & \pi \models_k^i \varphi \text{ or } \pi \models_k^i \psi \\
\pi \models_k^i X\varphi & \Leftrightarrow & i < k \text{ and } \pi \models_k^{i+1} \varphi \\
\pi \models_k^i B\varphi & \Leftrightarrow & 0 < i \leq k \text{ and } \pi \models_k^{i-1} \varphi \\
\pi \models_k^i \varphi U \psi & \Leftrightarrow & \exists h \cdot i \leq j \leq k \cdot \pi \models_k^i \psi \text{ and} \\
 & & \forall n \cdot i \leq n < j \cdot \pi \models_k^n \varphi \\
\pi \models_k^i \psi R \varphi & \Leftrightarrow & \forall j \cdot i \leq j < k \cdot \pi \models_k^j \varphi \text{ or} \\
 & & \exists n \cdot i \leq n \leq j \cdot \pi \models_k^n \psi
\end{array}$$

Figure 6: Kripke structures for SQGs of Figure 5.



We say that $K \models_k \varphi$ if for all paths π in K, $\pi \models_k \varphi$.

Our until and release operators U and R are bounded versions of the LTL operators U and R. Similarly, the *globally* operator G, defined as $G\varphi \equiv \bot\, R\varphi$, is a bounded version of LTL's G operator. The *finally* operator F is defined as usual as $F\varphi \equiv \top\, U\varphi$. In general, the unbounded LTL version of G is assumed to not hold. For example, $G\varphi$ does not hold as could be violated at $k + 1$; to decide $G\varphi$ in LTL with respect to a bound k, one needs a more sophisticated analysis of the KS K, as discussed in Biere[3].

To illustrate LSSL, consider a k-unfolding of the KS of Figure 6(a), and assume the distributions labeling the states correspond to mode s. Then, this KS has a path π such that $\pi \models_k G\,(P[D = \mathrm{s}] = 2/3)$ holds: the path that always returns to x. To automatically find π, we can model check the negation of this formula; as discussed in Section 5, π will be returned as a counterexample. Using the techniques in Biere[3], one can show that π also satisfies the unbounded LTL version of the formula.

5. MODEL CHECKING AND LEARNING

Bounded Model Checking: Given a KS K, LSSL formula φ, and bound k, a *bounded model checker* (BMC) efficiently verifies if $K \models_k \varphi$. If not, it returns one or more paths π in K that violate φ (i.e., counterexamples); otherwise, it returns true. Intuitively, a BMC applies the LSSL semantics inductively defined in Section 4 to each path π in K. We have implemented a simple prototype BMC for KSSs derived from SQTs and LSSL formulae, which first enumerates all paths in a KS and then for each path applies the LSSL semantics. This approach is efficient enough to check within milliseconds the onset of spirals. We are currently improving our handling of safety formulae (those without the F operator) by pruning, during SQT traversal, all subtrees of a vertex as soon as we detect that the current path satisfies φ. A more ambitious SAT-based BMC is also under development.

Machine Learning: Writing the LTL formulae that a reactive system should satisfy is a nontrivial task. Developers often find it difficult to specify system properties of interest. The classification of LTL formulae into *safety* (something bad should never happen) and *liveness* properties (something good should eventually happen) provides some guidance, but the task remains challenging.

Writing LSSL formulae describing emerging properties of CLHA networks is even more difficult. For example, what is the LSSL formula for spiral onset? In the following, we describe a surprisingly simple, machine-learning-based approach that we have successfully applied to spiral detection. The main idea is to cast the onset property as follows: *Is there a path in the given* SQT *leading to the core of a spiral?*, where the core of a spiral is the central point from which the spiral emanates, getting progressively farther away as it revolves around the point.

The implementation of our approach is simple as well. For an SEF (a 400×400 array of AP values) produced by the CellExcite simulator (see Figure 1), our Emerald tool set allows the user to select a path through the SEF's corresponding SQT simply by clicking on a point in the SEF, e.g. in the core of a spiral. If no spiral is present, the SQT path with maximum PMF (probability mass function) is returned. Note that this method is not restricted to spirals: path selection via clicking on a representative point can be applied to normal wave propagation, wave collision, etc.

The paths so obtained are then used to learn the LSSL formula for the property in question, such as spiral onset. The learning algorithm works as follows: (1) For each path of length k, where k is the height of the SQT, we define k attributes $a_1,\dots, a_k$ such that each a_i holds the PMF value of vertex v_i, for the mode we are interested in (for spirals, mode s). (2) Each path is classified by Emerald as spiral or nonspiral depending on whether or not the user clicked on a point (core); the classification is stored as an additional classifier attribute c. (3) All records $(a_i,\dots, a_k, c)$ are stored in a table, which is provided to the data-classification phase. (4) At the end of this phase, we obtain a path classifier which we translate into an LSSL formula.

Data classification[22] is generally a two-step process: training and testing. For *training*, we choose a classification algorithm that learns a set of descriptions of our training data set. The form of these descriptions depends on the type of classification algorithm employed. For *testing*, we use a test data set, disjoint from the training set and containing the class attribute with a known value. The *accuracy* of the classifier on a given test set is the percentage of the test records that are correctly classified. Various techniques can be used to obtain test and training sets from an initial set of records, such as X-Cross Validation.[8]

For classification purposes, we use a *descriptive classifier* (DC), which returns a set of if–then rules called *discriminant rules*. Underlying DCs are decision trees, rough sets, classification-by-association analysis, etc. A rule r has the form

$$(\bigvee_{i\in 1} a_i = v_i) \Rightarrow (c = v)$$

where I is a subset of $[1..k]$. Usually, each class c has an associated set of rules $r_1, \dots, r_n$; i.e., c is characterized by $\bigwedge_{i=1}^{n} r_i$. Using boolean arithmetic, this is equivalent to

$$(\bigvee_{i=1}^{n} \bigwedge_{j\in I_i} a_{ij} = v_{ij}) \Rightarrow (c = v)$$

The antecedent formula $\bigvee_{i=1}^{n}\bigwedge_{j\in I_i} a_{ij} = v_{ij}$ is called the *class-description formula* of the class c.

As is customary, we built a classifier for one class only (the class c), called the *target class*, using all other classes as one contrasting class. Hence the classifier consists of only one class-description formula, describing the target class. We say that we *learned* that formula. We have used Weka's decision-tree algorithm, but any other rule-based algorithm could have been used as well. The classifier we have learned for spirals is as follows:

```
if a7 <= 0.875 then
  if a2 <= 0.048 then ~c else c
else if a3 <= 0.078 then
  if a0 <= 0.025 then ~c else c
else ~c
```

Its translation into LSSL, where $\mathbf{X}^k$ stands for k repetitions of $\mathbf{X}$, generated the following formula:

$$X^2 P(D=s)>0.048 \wedge X^7 P(D=s)\leq 0.875 \vee$$
$$P(D=s)>0.025 \wedge X^3 P(D=s)\leq 0.078 \wedge X^7 P(D=s)>0.875$$

This formula is an approximate description of a spiral which we use together with Emerald's BMC to detect spiral onset within milliseconds. In case the BMC returned a false positive, we add the corresponding record to the classification table as part of a retraining phase; see Figure 1.

6. IMPLEMENTATION

Our techniques of Sections 2–5 have been implemented as the Emerald tool suite of the EHA environment. Emerald is a Java application that can be used to learn an LSSL formula for a particular spatial pattern, and to check the formula against a set of images (of the kind pictured on the right-hand side of Figure 3) that reproduce the discrete behavior of a CLHA network. For ease of use, Emerald provides two graphical tools, one for *Preprocessing* (classification) and the other for *Bounded Model Checking*.

The Preprocessing tool enables users to browse the various collections of images they have assembled for machine-learning purposes, and to view their SQT representation. The user can select a path leading to a spiral core by clicking on an appropriate stimulated point (in yellow) of the image. If the image does not contain a spiral, the user can choose the maximum PMF path or a generic stimulated point. Each path selected is stored in a data table in the form of the PMF sequence of stimulated modes in each node of the traversed SQT. All such paths are subsequently exported to Weka in a common format. Presently, we have customized Emerald for spiral detection, but we plan to extend the tool with the capability to classify any generic spatial pattern.

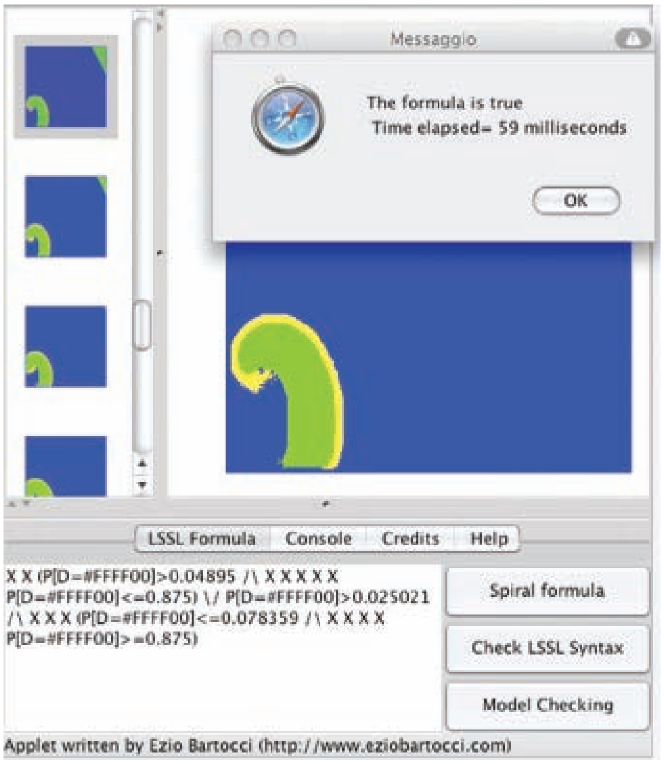
The BMC applet (Figure 7) enables the user to check an LSSL formula against the SQT representation of a specific image. As discussed in Section 5, the LSSL formula encodes the classifier for the spatial pattern under investigation. If the SQT in question fails to satisfy the formula, the resulting counterexamples (spirals) are reported to the user both as rows in the counterexample table and as red dots marking the core of the spiral contained in the image.

Table 1 contains our preliminary experimental results. For training and testing purposes, we used two different sets of images, each containing spirals and normal wave propagation. The first set of images was used to train the classifier; we supervised the training by discriminating between paths leading to a spiral core versus those (of maximum PMF) belonging to images that did not contain a spiral. From this first set we extracted 512 possible paths, and used Weka to build a ruled-based classifier with a very high prediction accuracy (99.25%).

The test set was divided into increasingly larger sets of images: 500, 550, 600, and 650 images. Applying the rule-based classifier on the first 500 images produced 67 wrongly classified paths. We used these paths to obtain a new, retrained classifier. We then used both classifiers on the remaining sets of images, and for each classifier and test set we computed the LSSL *formula accuracy*, as an estimate of how well the formula specifies the spatial pattern. As Table 1 shows, retraining considerably improves accuracy, and can be repeated each time a false classification is returned.

Weka's decision-tree algorithm took less than 9 s to construct a rule-based classifier from the training (512 records) and retraining (579 records) tables, respectively. Our model checker took between 1.67 and 7.09s, with an average of 4.72s to model check the SQT for a 400 × 400 SEF if no spiral was present, and between 1 ms and 4.64s, with an average of 230ms otherwise. All results were obtained on a PC equipped with a Centrino 2GHz processor with 1.5GB RAM.

Figure 7: EMERALD bounded model checker.



The formula is true
Time elapsed= 59 milliseconds

```
X X (P[D=#FFFF00]>0.04895 /\ X X X X X
P[D=#FFFF00]<=0.875) \/ P[D=#FFFF00]>0.025021
/\ X X X (P[D=#FFFF00]<=0.078359 /\ X X X X
P[D=#FFFF00]>=0.875)
```

Applet written by Ezio Bartocci (http://www.eziobartocci.com)

7. RELATED WORK

The use of hybrid automata to model and analyze spatial networks is a relatively new subject area, and includes application to Delta-Notch signaling networks,[9] coordinated control of autonomous underwater vehicles,[19] and aircraft trajectories and landing protocols.[7, 21] In contrast, our focus is on emergent behavior (in the form of spiral waves) in networks of cardiac myocytes, and the use of spatial superposition as an abstraction mechanism. Predicting spirals[4] in pure continuous models[23] is a more complicated process than what is implemented in EMERALD, where discrete SQT structures, obtained via mode abstraction and superposition, are used. Several logics have recently been proposed for describing the behavior and spatial structure of concurrent systems,[5, 6] and for reasoning about the topological aspects of modal logics and Kripke structures.[1] Unlike LSSL, these logics are not based on an abstraction mechanism like spatial-superposition that can be used to alleviate state explosion during model checking.

8. CONCLUSION

In this paper, we have presented a framework for specifying and detecting emergent behavior in networks of cardiac myocytes. Our approach, which uses hybrid automata, discrete mode abstraction, and bounded model checking, is based on a novel notion of spatial superposition and its related logic LSSL, and a new method for the automated learning of formulae in this logic from the spatial patterns under investigation. Our framework has been fully implemented in the EMERALD tool suite. Our preliminary experimental results are very encouraging, with a prediction accuracy of over 93% on a test set comprising 650 images. As future work, we plan to extend our framework to the learning of branching-time spatial-superposition properties and the more intricate problem of specifying and detecting spatiotemporal emergent behavior.

We also experimented with the SIFT (Scale-Invariant Feature Transform) algorithm, which detects and matches interesting features in images while preserving invariance constraints for scaling, translation, and rotation.[15] We found that SIFT performed matching well on images of spirals that were related to one another through rigid transformations. It was less successful, due to an insufficient number of matching keypoints, on spirals with more markedly different shapes. Also, SIFT and other image-processing techniques tend to process the entire image. Our approach, in contrast, uses logical formulae over SQT *paths* and densities of a particular CLHA mode (stimulated) along such paths.

Table 1: Experimental results.

Path Classifier	Test Set (550)	Test Set (600)	Test Set (650)
Trained (512 paths)	87.00%	88.83%	88.23%
Retrained (512 paths + 67 counterexamples)	97.10%	97.33%	93.07%

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. Research supported in part by NSF awards CCR-0133583, CNS-0509230, and CCF-0523863.

References

1. Aiello, M., Benthem, J., and Bezhanishvili, G. Reasoning about space: The modal way. *J. Log. Comput. 13*, 6 (2003), 889–920.
2. Bartocci, E., Corradini, F., Entcheva, E., Grosu, R., and Smolka, S.A. CellExcite: An efficient simulation environment for excitable cells. *BMC Bioinformatics*, *9*, Suppl 2 (2008), S3.
3. Biere, A., Cimatti, A., Clarke, E., Strichman, O., and Zhu, Y. Bounded model checking. In *Adv. in Comp. vol. 58: Highly Depend. Software*. Acad. Press, 2003.
4. Bray, M.A., Lin, S.F., Aliev, R.R., Roth, B.J., and Wikswo, J.P.J. Experimental and theoretical analysis of phase singularity dynamics in cardiac tissue. *J. Cardiovasc. Electrophysiol. 12*, 6 (2001), 716–722.
5. Caires, L., and Cardelli, L. A spatial logic for concurrency (part I). *Inf. Comput. 186*, 2 (2003), 194–235.
6. Caires, L., and Cardelli, L. A spatial logic for concurrency (part II). *Theor. Comput. Sci. 322*, 3 (2004), 517–565.
7. deOliveira, I., and Cugnasca, P. Checking safe trajectories of aircraft using hybrid automata. In *Proceedings of SAFECOMP 2002*. Springer-Verlag, Sept. 2002.
8. Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B. Witten, I.H., and Trigg, L. WEKA: A machine learning workbench for data mining. In *The Data Mining and Knowledge Discovery Handbook*, 1305–1314. Springer, 2005.
9. Ghosh, R., Tiwari, A., and Tomlin, C. Automated symbolic reachability analysis: with application to Delta-Notch signaling automata. In *HSCC*, 233–248, 2003.
10. Grosu, R., Bartocci, E., Corradini, F., Entcheva, E., Smolka, S.A., and Ye, P. EHA: An environment for the specification, simulation, analysis and control of networks of excitable hybrid automata. http://www.cs.sunysb.edu/~eha, 2009.
11. Grosu, R., Mitra, S., Ye, P., Entcheva, E., Ramakrishnan, I.V., and Smolka, S.A. Learning cycle-linear hybrid automata for excitable cells. In *Proceedings of HSCC'07, the 10th International Conference on Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, Pisa, Italy, April 2007. Springer Verlag, 245–258.
12. Henzinger, T.A. The theory of hybrid automata. In *Proceedings of 11th IEEE Symposium on Logic in Computer Science*, 1996, 278–293.
13. Hodgkin, A.L. and Huxley, A.F. A quantitative description of membrane currents and its application to conduction and excitation in nerve. *J. Physiol.*, *117* (1952), 500–544.
14. Kwon, Y., and Agha, G. Scalable modeling and performance evaluation of wireless sensor networks. In *IEEE RT Tech. and App. Symp.*, 2006, 49–58.
15. Lowe, D.G. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision 2*, 1999, 1150–1157.
16. Lu, Y. Concept hierarchy in data mining: Specification, generation and implementation. Master's thesis, Simon Fraser University, Dec. 1997.
17. Luo, C.H., and Rudy, Y. A dymanic model of the cardiac ventricular action potential: I. simulations of ionic currents and concentration changes. *Circ. Res.*, *74* (1994), 1071–1096.
18. Manna, Z., and Pnueli, A. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.
19. Pereira, F., and deSousa, J. Coordinated control of networked vehicles: An autonomous underwater system. *Aut. Remote Ctrl. 65*, 7 (2004), 1037–1045.
20. Shusterman, E., and Feder, M. Image compression via improved quadtree decomposition algorithms. *IEEE Trans. Image Processing 3*, 2 (Mar. 1994), 207–215.
21. Umeno, S., and Lynch, N. Safety verification of an aircraft landing protocol: A refinement approach. In *Proceedings of HSCC 2007*, Apr. 2007.
22. Wasilewska, A., and Ruiz, E.M. A classification model: Syntax and semantics for classification. In *RSFDGrC (2)*, 2005, 59–68.
23. Wedge, N.A., Branicky, M.S., and Cavusoglu, M.C. Computationally efficient cardiac biolectricity models toward whole-heart simulation. In *Proceedings of International Conference on IEEE Engineering in Medicine and Biology Society*, 2004, 1–4.
24. Ye, P., Entcheva, E., Grosu, R., and Smolka, S. Efficient modeling of excitable cells using hybrid automata. In *Proceedings of CMSB'05, the Third Workshop on Computational Methods in Systems Biology*, Edinburgh, Scotland, April 2005, 216–227.
25. Ye, P., Entcheva, E., Smolka, S.A., and Grosu, R. Modeling excitable cells using cycle-linear hybrid automata. *IET Syst. Biol.*, *2* (Jan. 2008), 24–32.

Radu Grosu (grosu@cs.sunysb.edu), Department of Computer Science, Stony Brook University, Stony Brook, NY.

Scott A. Smolka (sas@cs.sunysb.edu), Department of Computer Science, Stony Brook University, Stony Brook, NY.

Flavio Corradini (flavio.corradini@unicam.it), Department of Mathematics and Computer Science, University of Camerino, Camerino (MC), Italy.

Anita Wasilewska (anita@cs.sunysb.edu), Department of Computer Science, Stony Brook University, Stony Brook, NY.

Emilia Entcheva (emilia.entcheva@sunysb.edu), Department of Biomedical Engineering, Stony Brook University, Stony Brook, NY.

Ezio Bartocci (ezio.bartocci@unicam.it), Department of Mathematics and Computer Science, University of Camerino, Camerino (MC), Italy. Currently visiting the Department of Computer Science, Stony Brook University, Stony Brook, NY.

California State University
Department of Computer Science and Engineering

The Department of Computer Science and Engineering invites applications for a tenure rack position at the Assistant Professor level. Candidates must have a Ph.D. in Computer or Electrical Engineering, or a closely related field. The position is primarily to support the new B.S. in Computer Engineering program. The program has strong support from local industry and government entities. In addition, the Department offers the degrees B.S. in Computer Science (ABET accredited), B.A. in Computer Systems, B.S. in Bioinformatics and M.S. in Computer Science. Women and underrepresented minorities are strongly encouraged to apply. For the complete ad please visit http://cse.csusb.edu.

DEADLINE AND APPLICATION PROCESS: Applicants should submit a curriculum vitae, statement of teaching philosophy, description of research interests, an official copy of most recent transcripts, and have three letters of recommendation sent separately. Review of applications will begin January 15, 2009 and will continue until the position is filled; the position will start in September 2009. Please send all materials to:

Dr. George M. Georgiou, Chair
Department of Computer Science and Engineering
California State University, San Bernardino
5500 University Parkway
San Bernardino CA 92407-9393

Gonzaga University
Assistant or Associate Professor of Computer Science

Gonzaga University seeks applicants for an Assistant or Associate Professor of Computer Science. This is a full-time, tenure track position to begin in the fall semester, 2009. Required qualifications: demonstrated expertise in data mining, database management systems, scientific visualization, or bioinformatics; Ph.D. in computer science or closely related field. The Department's facilities include a computational science lab with a 512 node cluster, and a sensor network and robotics lab. The department is housed in the newly completed Paccar Center for Applied Science.

Gonzaga, with 7000 students, is in the center of Spokane, Washington along the Spokane River. Research opportunities are available with the Pacific Northwest National Laboratories and many businesses in the area. Spokane, the health care center for the inland Northwest, has a metropolitan area population of 500,000. The city offers one of the finest four-season living environments in the Pacific Northwest, with five ski resorts, more than 60 lakes, and several national forests nearby.

Review of applications will begin 1/12/09. Applications will be accepted until the position is filled. Please send a letter, complete curriculum vita, a statement of research and teaching objectives, and the names, addresses, and telephone numbers of at least three references to: Paul De Palma, Chair, Department of Computer Science, Gonzaga University, Spokane, WA 99258-0026. Electronic submissions in pdf format are preferred and should be sent to: depalma@gonzaga.edu. Gonzaga is a Catholic, Jesuit and humanistic university interested in candidates who can contribute to its distinctive mission. The University is an AA/EEO employer and educator committed to diversity

Open Positions at INRIA for
Tenured and Tenure-track Research Scientists

INRIA is a French public research institute in information and communication science and technology. It is an outstanding and highly visible scientific organization, a major player in the European Research Area heavily involved in most of research and development programs. INRIA has eight research centers in Paris, Bordeaux, Grenoble, Lille, Nancy, Nice – Sophia Antipolis, Rennes and Saclay that host 160 project-teams in partnership with universities and other research organizations. INRIA focuses the activity of over 1100 researchers and faculty members, 1200 PhD students and about 1000 post-docs and engineers, on fundamental research at the best international level, as well as on development and transfer activities in the following computer science and applied mathematics areas:

- Modeling, simulation and optimization of complex dynamic systems
- Formal methods in programming secure and reliable computing systems
- Networks and ubiquitous information, computation and communication systems
- Vision and human-computer interaction modalities, virtual worlds and robotics
- Computational Engineering, Computational Sciences and Computational Medicine

In 2009, INRIA is opening over 40 new positions within its 8 research centers:

- Junior and senior level positions,
- Tenured and tenure-track positions,
- Research and joint faculty positions with universities

These positions cover all the above research areas.
INRIA centers provide outstanding scientific environments and excellent working conditions. The institute offers competitive salaries and social benefit programs. It welcomes applications from all nationalities; it will arrange if needed visa and working permits (also for the spouse). French schooling and social programs for families are well organized and highly regarded.

Calendar and detailed application information at:

http: www.inria.fr/travailler/index.en.html
email: Laura.Norcy@inria.fr

North Carolina Central University
Computer Science Faculty Position

The Department of Mathematics and Computer Science invites applications for tenure-track faculty positions in all areas beginning Fall 2009. A Ph.D. in computer science or related area is required. The successful candidate must have a commitment to the academic process, excellence in research, education, and service, and to diversity in the community. The candidate must have a desire to participate in student academic and thesis advising, and curriculum development. We are particularly interested in candidates with research interests in artificial intelligence, computer vision, computer graphics, grid computing, robotics, computational biology, software engineering, multimedia applications, networks, mobile computing, wireless sensor networks and security/cryptography.

The campus is located in the Research Triangle Area, an ideal location in NC with several universities and high-tech companies. Applications and inquiries should be sent to ruma@nccu.edu. Further information can be found at: http://boole.cs.nccu.edu/emp2009/employment.html

Departmental resources include extensive computing facilities of workstations, servers and personal computers with multimedia capabilities and specialized networks and devices. Faculty members have access to high performance computing platforms provided by the university and its partners.

SUNY College at Plattsburgh
Assistant Professor, Computer Science

The Computer Science Department of the State University of New York, College at Plattsburgh is seeking qualified applicants for a tenure track assistant professor position starting fall 2009. SUNY Plattsburgh is a public liberal arts college with approximately 80 computer science majors.

Qualifications: Ph.D. in Computer Science, or closely related field. ABD considered. Applicants with strengths in software engineering, systems programming, security, networks, or with industry experience will be highly regarded. We seek a candidate with the potential to become an excellent teacher, the ability to excite students about involvement in research and applied projects, with creative ideas concerning curriculum development, an interest in exploring connections with other disciplines and an active program of scholarship.

Responsibilities include: Supporting undergraduate degree programs both in Computer

NANYANG TECHNOLOGICAL UNIVERSITY

FACULTY POSITIONS IN COMPUTER COMMUNICATIONS / COMPUTER SCIENCE / COMPUTING SYSTEMS / INFORMATION SYSTEMS WITH SINGAPORE'S SCIENCE AND TECHNOLOGY'S UNIVERSITY, NANYANG TECHNOLOGICAL UNIVERSITY

A member of the College of Engineering, the School of Computer Engineering (SCE) originated from the School of Applied Science that was established in 1988. Recognizing the rapid growth in the information technology arena, SCE was formed in 2000. It offers undergraduate training leading to BEng (Hons) in Computer Engineering or Computer Science as well as full-time and part-time graduate training leading to MSc and PhD. SCE's strengths lie in constantly maintaining industrial relevance in its training of undergraduate and graduate students, as well as pioneering innovative cutting-edge research. The school comprises of four divisions, namely Division of Computer Communications (CCM), Division of Computer Science (CSC), Division of Computing Systems (CPS) and Division of Information Systems (IS).

Applications are invited for appointment as **Associate Professor (A/P)** or **Assistant Professor (Ast/P)** in one of the four divisions. High-calibre applicants who possess a PhD and with a proven track record in research and teaching at university level are invited to apply for suitable appointments in the following areas:

- **Network System, Seamless Communications, Wireless Sensor Network (Ast/P in CCM)**
- **Multimedia Understanding, Mobile Multimedia (Ast/P in CCM)**
- **Distributed Systems, Collaborative Computing, or High Performance Computing (Ast/P in CSC)**
- **Modelling & Simulation (Ast/P in CSC)**
- **Embedded Systems (Ast/P in CPS)**
- **Intelligent Robotics & Control (Ast/P in CPS)**
- **Audio, Speech & Signal Processing (Ast/P in CPS)**
- **Information Retrieval/ Text Mining (A/P or Ast/P in IS)**
- **Agents and Services Computing (A/P or Ast/P in IS)**

Candidates for appointment at an Associate Professor level must possess an outstanding track record of research through publication in top rank journals, obtaining grants and academic leadership, as well as a willingness and demonstrated ability to teach at the undergraduate and graduate levels. Candidates for appointment at Assistant Professor level must demonstrate strong research potential and a willingness and ability to teach at the undergraduate and graduate levels.

The successful candidates are expected to carry out research in one of the research centres hosted by the school. Candidates are expected to teach on both MSc programmes and BEng Computer Engineering/Computer Science degrees offered by the school.

Further information about the school can be obtained at **http://www.ntu.edu.sg/sce**. Informal enquiries and submission of application forms can be made to **VD-SCE-ACAD@ntu.edu.sg**. Guidelines for application submission and application forms can be obtained from **http://www.ntu.edu.sg/ohr/Career/SubmitApplications/Pages/default.aspx**.

Closing Date: 15 April 2009

www.ntu.edu.sg

Postdoc Positions Available in Amsterdam

The Computer Systems section of the Department of Computer Science at the Vrije Universiteit is looking for two postdocs to work in the group of Prof. Andrew Tanenbaum. The theme running through all our research is how to design and build dependable and secure systems software. This research is being funded in part by a European Research Council Advanced Grant of €2.5 million awarded to Prof. Tanenbaum.

The goal of this project is to develop a highly dependable operating system that never goes down and which can repair itself and even upgrade itself automatically, without rebooting and without stopping running applications. The system will be based on MINIX 3 (www.minix3.org), which is a very small and reliable microkernel. The postdoc will do research in the area of dependable systems.

Another opening is for a 2-year postdoc in computer security. The candidate must have a strong publication record in the area of security with a preference for system security. The postdoc will carry out research within the project S-Mobile (www.cs.vu.nl/~ast/s-mobile) in the area of security of mobile platforms (e.g., Android).

For more information about both positions, please see

www.cs.vu.nl/~ast/jobs

vrije Universiteit amsterdam

Science and Information Technology; course instruction; and appropriate research and service activities.

SUNY College at Plattsburgh is an equal opportunity employer committed to excellence through diversity.

Salary: will be commensurate with qualifications, with excellent benefits. Review of applications will begin on March 1, 2009 and will continue until the position is filled. Original transcripts will be required prior to employment. Apply online at: https://jobs.plattsburgh.edu/applicants/Central?quickFind=50483 and include a cover letter, curriculum vitae, and three letters of reference.

Texas Tech University
Assistant/Associate Professor

Texas Tech University: The Department of Computer Science invites applications for a tenure-track position at the rank of Assistant / Associate Professor starting Fall of 2009. Preferences are given to candidates with background in networks. Other areas will be considered for excellent candidates. Applicants must have a Ph.D. degree in computer science or a closely related field. Successful candidates must have demonstrated achievements or potentials for excellence in research and teaching. The department of Computer Science is one of eight departments in the College of Engineering; it offers a PhD, MS, and BS in computer science and MS in Software Engineering. Faculty members perform scholarly and funded research in many areas, including artificial intelligence, databases, language theory, software engineering, data mining, robotics, and distributed and parallel computing. Texas Tech University is a comprehensive institution that includes law and medical schools with an enrollment of more than 28,000 students. Lubbock, a city of over 200,000, is a major economic and medical center on the Texas South Plains, with an excellent climate and numerous cultural opportunities. We offer competitive salaries, a friendly and cooperative environment, and excellent research facilities. Review will begin in January 2009 and continue until the position is filled. A letter of application, curriculum vitae, a summary of research and teaching goals, and three letters of reference should be submitted electronically at http://jobs.texastech.edu. Please use Requisition number 78066. Additional information is available at http://www.cs.ttu.edu. Texas Tech University is an equal opportunity/affirmative action employer and actively seeks applications from women, members of minority groups, disabled individuals, and veterans.

U.S. Air Force Academy

Department of Computer Science is accepting applications for our Coleman-Richardson Chair and Visiting Professor positions. See http://www.usafa.edu/df/dfcs/index.cfm or call (719) 333-3590 for details. U.S. Citizenship required.

University at Buffalo, The State University of New York
Faculty Positions in Computer Science and Engineering

The CSE Department invites excellent candidates in high performance computing and ubiquitous computing to apply for openings at the assistant professor level.

The department is affiliated with successful centers devoted to biometrics, bioinformatics, biomedical computing, cognitive science, document analysis and recognition, high performance computing, and information assurance.

Candidates are expected to have a Ph.D. in Computer Science/Engineering or related field by August 2009, with an excellent publication record and potential for developing a strong funded research program.

Applications should be submitted by March 15, 2009 electronically via recruit.cse.buffalo.edu.

The University at Buffalo is an Equal Opportunity Employer/Recruiter.

University of Denver
Professor and Department Chair

The Department of Computer Science (CS) at the University of Denver (DU) is seeking a dynamic and visionary individual from business or academia to lead the department during this expansion phase of the School of Engineering and Computer Science. Through its strategic plan-

Windows

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.

Universität Karlsruhe (TH)
Research University · founded 1825

KIT Karlsruhe Institute of Technology

The **Faculty of Informatics** at the University of Karlsruhe, one of the leading computer science departments in Germany, invites applications for a

Full Professorship with Tenure (W3) for Computer Graphics

The university is seeking an outstanding and innovative researcher from academia or industry. The successful applicant should have excellent qualifications and several years of experience in Computer Graphics.

A commitment to teaching and participation in informatics education is expected.

The University of Karlsruhe is an equal opportunities employer and encourages applications from women, minorities, and individuals with disabilities.

Applications including a resume, a list of publications indicating the five most important contributions, and a statement about future research (2 pages) should be sent by **30 March 2009** to:
Universität Karlsruhe (TH), Dean of the Faculty of Informatics, Am Fasanengarten 5, D-76131 Karlsruhe, Germany; email: dekanat@ira.uka.de.

ning, the faculty of our CS department have identified Software Engineering, Game Development, and Cyber Security as the key focus areas for the department. Our CS department benefits from a top quality faculty, strong partnership with industry, strong collaborations with other colleges within DU and internationally. The CS department offers degrees in both traditional and contemporary areas such as undergraduate degree in gaming, and graduate degree in Computer Science Systems Engineering. The primary focus of this new department chair will be on both educational and research programs at graduate and undergraduate levels. DU is a private university with a strong history of academic excellence, small classes, and emphasis on student engagement at all levels. DU is the oldest university in Colorado and its campus is located in the Denver metro area.

Individuals with a strong record of research, scholarship and excellence in teaching are encouraged to apply by sending their resume, statement of interest, and a list of five references to www.dujobs.org. PhD or PhD candidate in computer science or related areas and some level of leadership experience are required. The University of Denver is an AA/EOE.

University of Wisconsin-Madison
Assistant Professor

The Department of Computer Sciences at the University of Wisconsin-Madison has an opening for a tenure-track Assistant Professor, beginning August 2009.

We invite applications from outstanding candidates in all areas of Computer Science, and are especially interested in applications from candidates working in human computer interaction (HCI). Applicants should have a Ph.D. in computer science or a closely related field, and demonstrated strength in scholarly research. Successful candidates will be expected to teach at the undergraduate and graduate level, in addition to establishing a significant and highly-visible research program.

Applicants should submit a curriculum vita, a statement of research objectives and sample publications, and arrange to have at least three letters of reference sent directly to the department. Electronic submission of all application materials is preferred (see http://www.cs.wisc.edu/recruiting for details).

To ensure full consideration, applications, along with supporting material, should be received by March 15, 2009. Early submission is appreciated.

The UW-Madison is an equal opportunity/affirmative action employer and encourages women and minorities to apply. Unless confidentiality is requested in writing, information regarding the applicants must be released on request. Finalists cannot be guaranteed confidentiality. Employment may require a criminal background check.

For further information, send emails to recruiting@cs.wisc.edu .

TEXAS A&M UNIVERSITY

Dean
College of Architecture

Texas A&M University seeks applications for dean of the College of Architecture, one of the premier design research institutions in the world and the largest college of its kind in the nation.

The college is dedicated to generating knowledge and producing leaders in the fields of architecture, construction science, landscape architecture, urban planning and visualization.

The ideal candidate will share the college's united vision of significantly influencing the state of the art in the design, planning and construction of built and virtual environments and possess demonstrated ability to lead in a multi-disciplinary environment rich in resources.

Applications will be accepted through March 1, 2009. Details are available online:

http://deansearch.arch.tamu.edu/

Texas A&M University is an affirmative action, equal opportunity institution that is strongly and proactively committed to diversity.

EXECUTIVE DIRECTOR, CSAB Inc.

CSAB Inc. is accepting applications for the position of Executive Director. CSAB is a non-profit whose purpose is to advance the quality of computing disciplines in the public interest. CSAB supports and guides ABET accreditation of undergraduate programs of study in computer science, information systems, information technology, software engineering, and computer engineering.

The Executive Director reports to a seven-person volunteer Board of Directors. The Executive Director works with the Board to achieve CSAB goals including recruiting and managing the activities of approximately 200 volunteer program evaluators.

Job Summary: The Executive Director is the Chief Operating Officer of CSAB Incorporated. The Executive Director is responsible for the consistent achievement of CSAB's mission and financial objectives. The job entails a wide range of duties from clerical to managerial.

This is a part time position in which the workload varies depending on the schedule of Board activities. Some travel is required. Geographic location is negotiable. Salary is negotiable.

Additional information and application information may be found at

www.csab.org

[CONTINUED FROM P. 112] are dismissed as "The Rest"; the poor dears, they seem to just keep falling farther and farther behind.

Everyone in your daughter's law school takes it as a matter of course that the law they are studying is changing to match the new enhancements. The law will be upgraded, the Enhanceds believe, just as they get new physical and mental upgrades every time they go home. In fact, the paper your daughter is working on over the holidays concerns whether a Natural can truly enter into an informed relationship with an Enhanced, even for something as innocuous as a date.

We are at a turning point in human history. Today, for the first time in hundreds of thousands of years, our technologies are not only aimed outward at modifying our environment. Rather, the GRIN technologies—the genetic, robotic, information, and nano processes, all based on computing technologies evolving at the pace of Moore's Law if not faster—increasingly aim inward at changing who we are and what we can be. Not in some distant future but right now, on our watch.

How might such radical evolution influence what it means to be human? Talk to those deploying the GRIN technologies and you hear three scenarios—Heaven, Hell, and Prevail.

In "Heaven," we conquer pain, suffering, ignorance, stupidity—even death—in a perfection of the human condition. In it, traditional definitions of humanity are increasingly remote. There are few divisions like those in your daughter's law school scenario because it's so difficult to remember why anyone would want to cling to Version 1.0 humanity. Being a knowledge-based creature is far preferable to being what Ray Kurzweil calls "mostly original substrate humans."

In "Hell," pessimists see a mirror-image curve in which the power of the GRIN technologies inevitably gets into the hands of madmen or fools, leading to disaster for all. If conflict between different species of humans doesn't get us, then the genetically engineered microbes carefully designed to be 100% fatal or the self-replicating energy-devouring nanobots will. The outcomes are the same—annihilation of the human race within 20 years. Entirely too imaginable.

Even in the face of unprecedented threats, 'Prevail' reflects faith that the ragged human convoy of divergent perceptions, piqued honor, posturing, insecurity, and humor will wend its way to glory.

Both Heaven and Hell are technodeterministic, assuming that our gear shapes history. In neither is your daughter able to do much to shape her generation's future. The critical driver is the smooth curve of Moore's Law, measuring progress by the number of transistors we get to talk to one another.

As a humanist, however, I root for "Prevail," which is not some middle ground between Heaven and Hell. Way off in its own territory, it assumes what really matters is not how many transistors we connect but how many ornery, cussed, imaginative, unpredictable humans we connect. Its measure is not individuals bragging about their latest cognitive implants, leaving your daughter and others like her frightened and lonely, but something far larger, measured in *group* transformation.

How do we know which scenario we are entering? Heaven and Hell both have the virtue of being obvious. We see bellwethers in the headlines every day. But suppose you see second-order *network* effects—group effects. Could they be early warnings of Prevail? Suppose you've seen cellphones going from curiosity to commonplace in 30 years. There are now more than one of them for every two humans on earth—the fastest uptake of any technology in history, including the polio vaccine. Suppose as a result you see some of the greatest economic and social transformations in some of the most unexpected places, from Bangladesh to Nigeria. I am, of course, describing the present.

If the harbingers of Prevail are the appearance of many collaborative, bottom-up, worldwide human solutions, what do they say about eBay? Not just the world's biggest flea market but a network of millions of people producing highly complex solutions without leaders. Facebook causes us to reconsider the meaning of such a basic human institution as "friend." YouTube recently helped shape the most interesting election in a lifetime. And what about Twitter?

Prevail embraces uncertainty. Even in the face of unprecedented threats, it reflects faith that the ragged human convoy of divergent perceptions, piqued honor, posturing, insecurity, and humor will wend its way to glory. The embedded assumption is that even if a smooth curve of exponential change describes the future of technology, it will not map onto the messy world of human fortunes.

Prevail is driven by faith in human cussedness, based on a hunch that you can count on humans to throw The Curve of exponential change a curve of their own. It is also a belief that transcendence resulting from humans taking control of their own evolution is unlikely to be part of any simple scheme.

The significance of all this can hardly be understated. Despite the billions of galaxies, each with billions of stars, we cannot detect any other life in the universe. Why not? Perhaps every intelligent species eventually takes control of its own evolution. Maybe such radical evolution is the final exam. Maybe everyone else has already flunked.

Let's not flunk, too.

Joel Garreau (www.garreau.com) is the author of *Radical Evolution: The Promise and Peril of Enhancing Our Minds, Our Bodies—and What It Means to Be Human.*

DOI:10.1145/1467247.1467272

Peter Winkler

Puzzled
Solutions and Sources

Last month (February 2009, p. 104) we posed a trio of brain teasers concerning algorithm termination. Here, we offer some possible solutions. How did you do?

1. Pentagon Problem

Solution. This puzzle first appeared, as far as I know, at the International Mathematics Olympiad of 1986. Known today as "the pentagon problem," it generated enormous interest and a variety of imaginative solutions. I present one of them here, due independently to at least two people, one of whom is Bernard Chazelle, a computer scientist at Princeton University. Let x_0, x_1, x_2, x_3, and x_4 be the five numbers, summing to $s > 0$, with indices taken modulo 5. Define a doubly infinite sequence z by $z_0 = 0$ and $z_i = z_{i-1} + x_i$. The sequence z is not periodic but is periodically ascending; $z_{i+5} = z_i + s$. In the example, the x values are 2,4,−3,1,−3; $s = 1$; and the z sequence is

... −2 0 4 1 2 −1 1 5 2 3 0 2 6 3 4 1 3 7 4 5 2 4 8 5 6 ...

where the rightmost 0 represents z_0.

If x_i is negative, $z_i < z_i{-}1$ and flipping x_i has the effect of switching z_i with $z_i{-}1$ they are now in ascending order. Simultaneously, it does the same for all pairs z_j, $z_j{-}1$ whose indices are shifted from them by multiples of five. Thus, flipping labels amounts to sorting z by adjacent transpositions.

Tracking the progress of the sorting process needs a potential function Φ to measure the degree to which z is out of order. Let i^+ be the number of indices $j > i$ for which $z_j < z_i$; note i^+ is finite and depends only on i modulo 5. We let Φ be the sum $0^+ + 1^+ + 2^+ + 3^+ + 4^+$. In the example, 0^+ is 0 (since there are no entries smaller than 0 to the right of z_0), 1^+ is 1, 2^+ is 13, 3^+ is 2, and 4^+ is 4, for a total of 20.

When x_{i+1} is flipped, i^+ decreases by one, and every other j^+ is unchanged, so Φ decreases by 1. When Φ hits 0 the sequence is fully sorted so all labels are non-negative and the process must terminate. In the example, 20 steps later the sequence has turned into

... 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 ...

where the first 3 is the z_0 term, and thus the numbers around the pentagon are now 0, 0, 0, 0, 1.

We conclude that the process terminated in exactly the same number (the initial value of Φ) of steps regardless of choices, and the final configuration is independent of choices. The reason is there is only one sorted version of z. Moreover, the proof works with 5 replaced by any integer greater than 2.

2. Billiard Balls

Solution. If you've played with this puzzle you've found that the algorithm does seem to terminate, no matter where you start or what you do, but can take rather a long time to do so. One might hope that (as in Puzzle 1) some non-negative-integer-valued "potential function" must go down at each step, proving that the algorithm must terminate, though there seems to be no simple one here. For example, you might have noticed that the sum of the distances between each billiard ball's current position and where it belongs cannot go up; alas, it may not go down either.

The following argument was devised by Noam Elkies, a mathematician at Harvard University. The ball that is deliberately moved to its correct position in a given step is said to be "placed." Suppose there is an infinite sequence of steps. Then, since there are only finitely many possible states (permutations), there must be a cycle; so, let ball k be the highest-numbered ball placed *upward* in the cycle. (If no ball is placed upward, the lowest-number ball placed downward is used in a symmetric argument). Once ball k is placed, it can be dislodged upward and placed downward again, but nothing can ever push it below position k. Hence it can never again be placed upward—a contradiction.

In fact, the algorithm always terminates in at most $2^{n-1} - 1$ steps, but there are more than exponentially many permutations that can take exactly the maximum number of steps to sort. These and other results, plus some history of this horribly inefficient sorting algorithm can be found at math.dartmouth.edu/˜pw/papers/sort.pdf. I am indebted to mathematician and writer Barry Cipra of Northfield, MN, for bringing the puzzle to my attention.

3. Notorious Collatz Conjecture

Readers interested in the rich history of this puzzle will appreciate a delicious survey article by Jeffrey C. Lagarias of the University of Michigan in *American Mathematical Monthly 92* (1985), 3–23; www.cecm.sfu.ca/organics/papers/lagarias/paper/html/paper.html.

All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

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.

Future Tense, one of the revolving features on this page, presents stories from the intersection of computational science and technological speculation, their boundaries limited only by our ability to imagine what will and could be.

DOI:10.1145/1467247.1467273

Joel Garreau

Future Tense
Radical Evolution

Technologies powerful enough to modify our minds, memories, metabolisms, personalities, and progeny are powerful enough to transform our own evolution.

IN 1913, THE U.S. Government prosecuted Lee De Forest for telling investors that his company, RCA, would soon be able to transmit the human voice across the Atlantic. This claim was so preposterous, prosecutors asserted, that he was obviously swindling potential investors. He was ultimately released, but not before being lectured by the judge to stop making any more fraudulent claims.

With this legal reasoning in mind, consider the scenarios I describe here. They are not predictions but meant to be credible portrayals of possible near-term futures, factually grounded in computer-enabled technologies, all unquestionably under development today.

Flash forward 15 years. Look at the girl who is today your second-grade daughter. Imagine she is just home for the holidays. You were so proud of her when she not only put herself through Ohio State but graduated summa cum laude. Now she has taken on her most formidable challenge yet: competing with her generation's elite in her fancy new law school. You want to hear all about it. But the difference between this touching tableau and those of the past is that in it, technologies designed to modify our minds, memories, metabolisms, personalities, progeny—indeed, what it means to be human—are now pouring onto the market. She is competing against all those with the will and wherewithal to adopt them.

"What are your classmates like, honey?," you say.

"They're all really, really smart," she says. How, she wonders, does she explain what the enhanced kids are like? She knows her parents have read about what's going on. But actually dealing with some of her new classmates is decidedly strange. These enhanced students have amazing thinking abilities. They're not only faster and more creative than anybody she's ever met but faster and more creative than anybody she's ever imagined. They have photographic memories and total recall. They devour books in minutes. They're also beautiful, physically.

They talk casually about living a long time, perhaps forever, always discussing their "next lives." One mentions how, after he makes his pile as a lawyer, he plans to be a glassblower, after which he wants to be a nanosurgeon.

What really matters is not how many transistors we connect but how many ornery, cussed, imaginative, unpredictable humans we connect.

Another fell while jogging, opening up a nasty gash on her knee. But instead of rushing to a hospital, she just stared at the wound, focusing her mind on it, triggering a metabolic cascade that caused the bleeding simply to stop. This same friend had been vaccinated against acute pain so she didn't feel it for long anyway.

They always seem to be connected to one another, sharing their thoughts no matter how far apart, with no apparent gear. They call it "silent messaging." It seems almost like telepathy. They have this odd habit of cocking their heads in a certain way whenever they want to access information, as if waiting for a wireless delivery to arrive; inevitably, it does. They don't sleep for a week or more at a time and joke about getting rid of the beds in their cramped dorm rooms.

They are unfailingly polite when your daughter can't keep up with their conversations, as if she were deficient in some way. They can't help but condescend, however, when she protests that embedded technology is not natural for humans.

They've nicknamed her "Natural," which is what they call all those who could be like them but choose not to be, referring to themselves as "Enhanced." Those with neither the education nor the money to consider keeping up with the exploding augmentation technologies [CONTINUED ON P. 110]

ACM CREATIVITY & COGNITION 2009

EVERYDAY CREATIVITY: SHARED LANGUAGES & COLLECTIVE ACTION

OCTOBER 27-30 2009

Berkeley Art Museum & UC Berkeley

CREATIVITY IS PRESENT IN ALL WE DO...

The 7th Creativity and Cognition Conference is to be held at the Berkeley Art Museum and the University of California, Berkeley (USA). The conference provides a forum for lively interdisciplinary debate exploring methods and tools to support creativity at the intersection of art and technology. We welcome submissions from academics and practitioners, makers and scientists, artists and theoreticians.

*FULL PAPERS *ART EXHIBITION *LIVE PERFORMANCES
*DEMONSTRATIONS *POSTERS *WORKSHOPS
*TUTORIALS *GRADUATE SYMPOSIUM

CALL FOR PARTICIPATION

KEYNOTE SPEAKERS

MIHÁLY CSÍKSZENTMIHÁLYI
PROFESSOR OF PSYCHOLOGY & MANAGEMENT
Claremont Graduate University **[CA, USA]**

JOANN KUCHERA-MORIN
DIRECTOR, ALLOSPHERE RESEARCH LABORATORY
California Nanosystems Institute **[CA, USA]**

everyday creativity

SUBMISSION BY: APRIL 24 FOR MORE INFO SEE: WWW.CREATIVITYANDCOGNITION09.ORG

CONFERENCE COMMITTEE General Chair **Nick Bryan-Kinns [UK]** Program Chair **Mark Gross [USA]** Program Co-Chair **Ron Wakkary [Canada]** Program Co-Chair **Hilary Johnson [UK]** Program Co-Chair **Jack OX [USA]** Demonstrations & Posters Chair **Yukari Nagai [Japan]** Tutorials Chair **Yusuf Pisan [Australia]** Workshops Chair **Ryohei Nakatsu [Singapore]** Graduate Student Symposium Co-Chair **John C Thomas [USA]** Graduate Student Symposium Co-Chair **Celine Latulipe [USA]** Art Exhibition Chair **Jennifer Sheridan [UK]** Art Exhibition Executive Committee **Sara Diamond [Canada]** Treasurer **Ellen Do [USA]** Sponsorship **Doug Riecken [USA]** Publicity **David A Shamma [USA]** Local Inspiration **Richard Rinehart [USA]** Local Committee Chair **Daniela Rosner [USA]** Steering **Ernest Edmonds [Australia]**

PRINTED PROCEEDINGS PUBLISHED BY ACM PRESS & APPEAR IN THE ACM DIGITAL LIBRARY

SIGCHI special interest group computer-human interaction

acm Association for Computing Machinery

The Best Place to Find the Perfect Job... Is Just a Click Away!

No need to get lost on commercial job boards.
The ACM Career & Job Center is tailored specifically for you.

JOBSEEKERS

- ❖ Manage your job search
- ❖ Access hundreds of corporate job postings
- ❖ Post an anonymous resume
- ❖ Advanced Job Alert system

EMPLOYERS

- ❖ Quickly post job openings
- ❖ Manage your online recruiting efforts
- ❖ Advanced resume searching capabilities
- ❖ Reach targeted & qualified candidates

NEVER LET A JOB OPPORTUNITY PASS YOU BY!
START YOUR JOB SEARCH TODAY!

http://www.acm.org/careercenter

Association for Computing Machinery

Advancing Computing as a Science & Profession

POWERED BY JOBTARGET