COMMUNICATIONS OF THE ACM

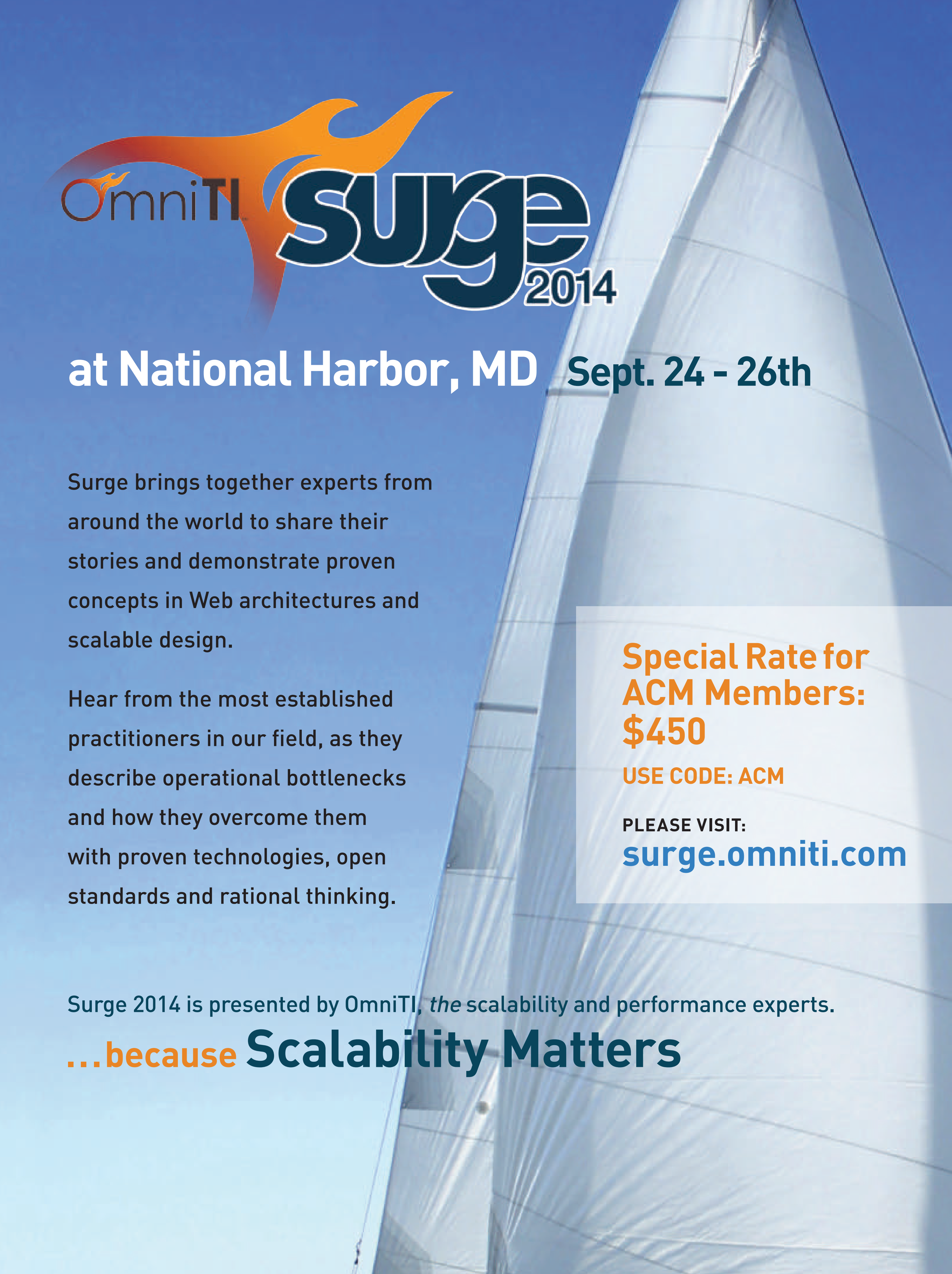
CACM.ACM.ORG

06/2014 VOL.57 NO.06

Leslie Lamport

Recipient of ACM's A.M. Turing Award

Association for Computing Machinery

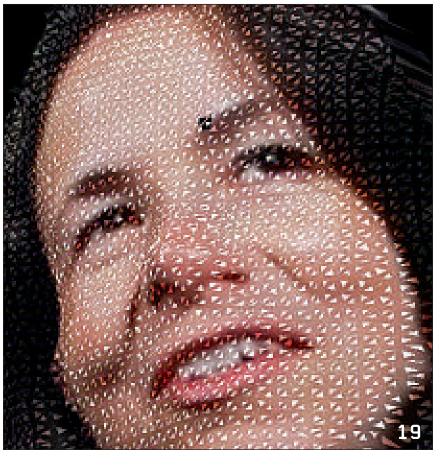


COMMUNICATIONS OF THE ACM

Departments

Last Byte

News



Viewpoints

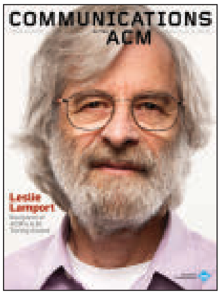
IMAGE BY JAMES GEORGE AND JONATHAN MINARD

06/2014
VOL. 57 NO. 06

Practice

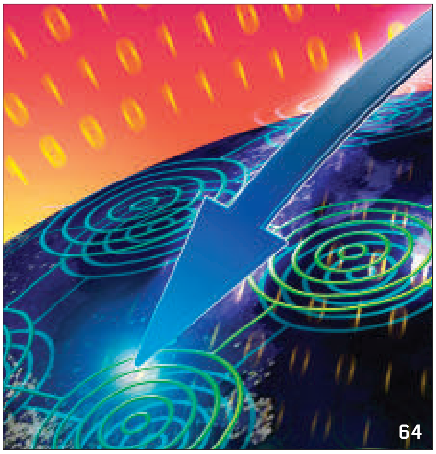
Articles' development led by acmqueue
queue.acm.org

IMAGES BY ALICIA KUBISTA, ROY WIEMANN, IGNOTUS THE MAGE

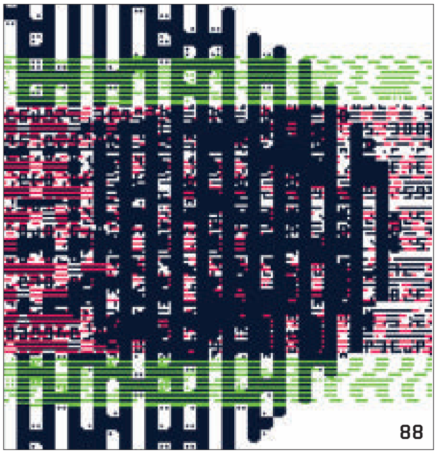


About the Cover:
Leslie Lamport, recipient of ACM's 2013 A.M. Turing Award, pictured on Microsoft's Silicon Valley campus by San Francisco-based photographer Richard Morgenstein. Lamport was cited for his fundamental contributions to the theory and practice of distributed and concurrent systems.

Contributed Articles



Review Articles



Research Highlights

Association for Computing Machinery
Advancing Computing as a Science & Profession

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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.

CONTACT POINTS

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

WEBSITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY
10121-0701
T (212) 626-0686
F (212) 869-0481

Director of Media Sales
Jennifer Ruzicka
jen.ruzicka@hq.acm.org

Media Kit acmmediasales@acm.org

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

Subscriptions
An annual subscription cost is included in ACM member dues of $99 ($40 of which is allocated to a subscription to *Communications*); for students, cost is included in $42 dues ($20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is $100.

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) 626-0686.

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

Printed in the U.S.A.

Association for
Computing Machinery

PLEASE RECYCLE THIS MAGAZINE

DOI:10.1145/2610445

Mehran Sahami and Steve Roach

Computer Science Curricula 2013 Released

For over 40 years, ACM and IEEE-Computer Society have sponsored international curricular guidelines for undergraduate programs in computing. The rapid evolution and expansion of the computing field and the growing number of topics in computer science have made regular revision of curricular recommendations necessary. The latest volume in the series, Computer Science Curricula 2013 (CS2013), was released last fall (http://dx.doi.org/10.1145/2534860).

The goal of CS2013 is to provide advice and guidance to the computing education community throughout the coming decade. The CS community was broadly consulted on the updates encompassed in the CS2013 curricular guidelines, with nearly 200 computer scientists from around the world contributing in some form to the final report. Presentations of preliminary versions of the report at a variety of conferences and workshops provided additional opportunities to engage the community throughout CS2013's evolution. By redefining the CS knowledge areas, rethinking the essentials necessary for a CS curriculum, and identifying working exemplars of courses and curricula, CS2013 attempts to balance the growth in the field with the need to keep recommendations realistic and implementable in the context of undergraduate education.

The high-level themes on which the CS2013 effort is based include:

▸ *The "Big Tent" view of CS.* As CS expands to include more multidisciplinary work and new programs of the form "Computational X" are developed, it is important to embrace an outward looking view that sees CS as a discipline actively seeking to work with and integrate into other disciplines.

▸ *Institutional needs.* Curricula exist in the context of specific institutional needs, goals, and resource constraints. Curricular guidelines must be relevant to a wide variety of institutions in the U.S. and internationally, including large and small, research and teaching, public and private, with programs ranging from two to four (or more) years. As a result, CS2013 provides guidance in developing curricular structure through a *tiered* set of core topics, where a small set of Tier 1 topics are considered essential for all CS programs, but individual programs have flexibility with regard to their coverage of Tier 2 and Elective topics.

▸ *Managing the size of the curriculum.* Although the CS field has expanded significantly in the past decade, it is simply not feasible to proportionately expand the size of the curriculum. CS2013 manages the size of curricula without requiring more instructional hours than the previous CC2001 (http://www.acm.org/sigcse/cc2001) and CS2008 (http://bit.ly/1h76wyx) guidelines by restructuring sets of knowledge areas where common themes are identified; making the expected depth of coverage of topics explicit, in part by providing example outcomes; and by changing topic emphasis within individual topic areas to reflect the state of the art and practice.

▸ *Actual course and curricular exemplars as opposed to stylized course guidance.* While the previous curricular effort, CC2001, took on the challenge of providing descriptions of stylized courses incorporating the knowledge units defined in that report, it was felt in retrospect that such course guidance did not have much impact on actual course design. CS2013 takes a different approach: identifying examples of actual fielded courses and curricula—from a variety of international universities and colleges—to illustrate how topics in the CS Body of Knowledge may be covered and combined in diverse ways. The final report contains over 80 course exemplars and five full curricular exemplars from a variety of institutions.

The CS2013 final report includes a complete update to the Body of Knowledge in CS, organized around 18 knowledge areas. Compared to CC2001 or CS2008, the Body of Knowledge has been substantially reorganized. A new knowledge area—Software Development Fundamentals—encompasses what was previously called Programming Fundamentals. It also includes a cohesive set of fundamental software development concepts, including topics in algorithms, design, programming, and software development processes. Systems Fundamentals is a new knowledge area that identifies common themes among operating systems, networking, and architecture. Extracting fundamental concepts, for example, caching, in one place encourages programs to rethink how these topics are presented throughout the curriculum. Additionally, recent developments in the field (such as the ubiquity of parallel computing and a need for better understanding of computer security) have given rise to the development of new knowledge areas in Parallel and Distributed Computing, Information Assurance and Security, and Platform-Based Development.

Topics receiving less required coverage in CS2013 relative to previous guidelines include digital logic, numerical methods, Web page construction, search engines, and language translation. Advanced coverage of many of these topics still appears in CS2013 as elective material.

CS2013 also outlines important characteristics of CS graduates beyond technical expertise in relation to professional practice (for example, communication skills, teamwork, and ethics) as components of an undergraduate experience. We believe this report will help the community in continuing to evolve computing curricula to be modern and relevant. The CS2013 Final Report, along with supporting materials such a spreadsheet to allow programs to map their curricula against the CS2013 learning outcomes, is available at http://cs2013.org. C

Mehran Sahami (sahami@cs.stanford.edu) and **Steve Roach** (sroach55@hotmail.com) served as co-chairs of the CS2013 Steering Committee.

DOI:10.1145/2611566

Vinton G. Cerf

The House Elves of ACM

This is my last column as President of ACM. It has been an honor to serve and I look forward to a continuing role as Past President until June 2016. It has been a privilege to take up a page of each edition of *Communications* for the past 24 months and I have enjoyed the discipline that such a commitment inspires. It has also been enormously helpful to have Diane Crawford on my case when the column is late—she is one of the many staff who serve not only the professional and student members of ACM but all the many others who benefit from attendance at our conferences and workshops and who have access to our published literature.

Those of you familiar with the *Harry Potter* series will recognize the reference to house elves. I do *not* mean to imply that any of our hard-working staff are mistreated (à la Dobby). The house elves keep the great houses running behind the scenes. I use the term in appreciation of the often invisible but absolutely vital role every employee of ACM plays in making our institution productive and smooth operation. As President, I have been deeply impressed by the leadership skills shown by all the ACM management, particularly the dynamic duo: John White (CEO) and Pat Ryan (COO). If you have served in some capacity on the executive committee, the council, as a SIG chair, or on one of the many ACM boards or committees, you will be similarly conscious of the amazing amount of work they manage.

We have a truly dedicated staff that can be found at http://www.acm.org/key-people/staff. While ACM is deeply dependent on the extraordinary contributions of our volunteers, the organizational framework and underlying support for this work is key to ACM's successful operation. So I want to make a point of thanking the ACM staff for all they have done and continue to do for all of us.

This issue of *Communications* is also our awards issue and on the cover you see the incomparable Leslie Lamport, the recipient of the **2013 ACM A.M. Turing Award** for his almost uncanny work on distributed systems and protocols. I have known Leslie and his work for a long time and it will be a great pleasure to present him with our highest honor at the ACM Awards Ceremonies in San Francisco on June 21.

Other honorees to be feted at the event include: David Blei, recipient of the **ACM-Infosys Foundation Award** for his strikingly scalable, statistical topic modeling work. And the team responsible for the development of Coq will receive the **Software System Award**: Thierry Coquand, Gérard Huet, Christine Paulin-Mohring, Bruno Barras, Jean-Christophe Filliâtre, Hugo Herbelin, Chet Murthy, Yves Bertot, and Pierre Castéran.

Pedro Felipe Felzenszwalb will receive the **2013 Grace Murray Hopper Award** for contributions to object recognition in pictures and video. Robert D. Blumofe and Charles E. Leiserson are recognized for their contributions to efficient and robust parallel computation with the **Paris Kanellakis Theory and Practice Award**. And Susan H. Rodger will receive the **Karl V. Karlstrom Outstanding Educator Award** for her work in developing CS education in grade schools.

ACM-W named Susan T. Dumais the **2014–2015 Athena Lecturer**. Dumais introduced novel algorithms and interfaces for interactive retrieval. For their long-standing work as series editors of the *Springer Lecture Notes in Computer Science*, Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen will receive the **Distinguished Service Award**. And Sanjam Garg will receive the **2013 Doctoral Dissertation Award**.

The **2014 Presidential Award** goes to Mehran Sahami for his exhaustive efforts to produce CS2013. And the **Outstanding Contribution of ACM Award** goes to two exceptional staffers from ACM HQ: Russell Harris and Donna Cappo.

I should also note that ACM awards were also presented last November at the SC13. The **ACM Gordon Bell Prize** was awarded to the researchers responsible for the 11-petaflop simulations: Diego Rossinelli, Babak Hejazialhosseini, Panagiotis Hadjidoukas, and Petros Koumoutsakos from ETH Zurich; Costas Bekas and Alessandro Curioni from IBM Zurich Research Laboratory; Adam Bertsch and Scott Futral from Lawrence Livermore National Laboratory; and Steffen Schmidt and Nikolaus Adams from Technical University Munich.

Jack Dongarra received the **ACM-IEEE CS Ken Kennedy Award** for his leadership in designing and promoting standards for mathematical software used to solve numerical problems common to high-performance computing (HPC). And Jonathan Lifflander and Edgar Solomonik received the **ACM-IEEE George Michael Memorial HPC Fellowships** for 2013 for their projects on algorithms for large-scale system and parallel physical equation solutions, respectively.

At the awards banquet in June we will also recognize the 2013 ACM Fellows. I hope to see many of you there.

Live long and prosper.

Vinton G. Cerf, ACM PRESIDENT

ASSETS 2014

The 16th International ACM SIGACCESS Conference
on Computers and Accessibility

October 20-22, Rochester, NY, USA

- Individuals with hearing, sight and other sensory impairments
- Individuals with motor impairments
- Individuals with memory, learning and cognitive impairments
- Individuals with multiple impairments
- Older adults with diverse capabilities
- Professionals who work with these populations

Request Mentors by
March 7, 2014

Long Paper
Submissions by
May 9, 2014

Poster, Demo, Doctoral
Consortium & Student
Research Competition
Submissions by
June 27, 2014

http://assets14.sigaccess.org/

DOI:10.1145/2617777

Efficient Code to Counter Dying Moore's Law

Moshe Y. Vardi's Editor's Letter "Moore's Law and the Sand-Heap Paradox" (May 2014) took me back to my computer engineering education in the 1970s, a period of transition from expensive to (relatively) inexpensive hardware. My classes, both theory and practice, required that I understand microcode and software execution environments well enough to avoid gross inefficiencies. If Moore's Law is indeed winding down, as Vardi said, software practitioners must focus even more than they already do on developing efficient code.

In my more than 35 years as a software professional, I have noted with dismay the bloatware phenomenon fueled by the expectation that Moore's Law would mask inefficient software. Developing resilient, secure, efficient software requires more skills, time, and money, along with a different mind-set, from what we see in the commercial software industry today.

No one should count on a breakthrough on the hardware side in the face of Moore's Law's impending demise, although I will be delighted if proven wrong. Software researchers and engineers alike (and the organizations funding them) must reset their expectations vis-à-vis hardware advances. As Vardi said, "new algorithms and systems" and better use of existing resources through virtualization and software parallelism can help mitigate the slowdown in hardware advances.

David K. Hemsath, Round Rock, TX

Deeper Roots of Transactional Programming

Vincent Gramoli's and Rachid Guerraoui's article "Democratizing Transactional Programming" (Jan. 2014) suggested transaction theory was first formalized in 1976. As with many early developments in computing, the concepts of transaction management for data storage and retrieval developed in engineering practice *before* emerging in research papers or industry standards. Even so, the article postponed the theorization of transactions by at least a couple of years. Its Figure 1 ("History of Transactions") pinpointed an article by Eswaran et al. (including Jim Gray) called "Notions of Consistency and Predicate Locks in a Database System" in *Communications* (Nov. 1976) as the earliest published work on transactions. In fact, Eswaran et al. had condensed their own substantively identical IBM Research technical report (Dec. 30, 1974) *On the Notions of Consistency and Predicate Locks in a Data Base System*.

Reflecting the considerable development of industrial data management systems by the early 1970s, at least some essential features of the transaction concept had already been presented in an earlier published work—the *CODASYL Data Base Task Group April 71 Report*—which was indeed the first industry standard in concurrent database management. It included the now ubiquitous terms "data definition language" and "data manipulation language" to enable the declaration of a schema (and views upon it) and specified the statements needed to retrieve, insert, and update data elements.

In 1974, Eswaran et al. wrote: "While fairly static locking schemes are acceptable in a conventional operating system, a particular data base transaction may lock an arbitrary logical subset of the data base." A transaction that establishes priority claims over access to a set of shared related data items enables execution of concurrent programs affecting consistent (application-meaningful) state transitions.

Although the CODASYL report did not include the term "transaction," it did include rules for managing locking and contention between different "run-units," or independent program executions accessing a single data store. The CODASYL data manipulation language verbs `OPEN` and `CLOSE` demarcated units of work in which record sets and individual records could be locked, and competing run-units allowed degrees of access or completely excluded until the run-unit closed. The CODASYL report also contemplated rollback semantics in the face of failures.

Maurice Wilkes, a famous early pioneer of computing, used the term "transaction" when describing "transaction journals" as recoverable records of "a condensed summary of the transaction recorded immediately before the update is made" in his 1972 article "On Preserving the Integrity of Data Bases" in *Computer Journal*. Two years later, R.A. Davenport's "Design of Transaction-Oriented Systems Employing a Transaction Monitor" in *Proceedings of the ACM Annual Conference* recalled the role of early transaction processing monitors in crystallizing the transaction abstraction.

Notwithstanding these efforts, Eswaran et al.'s 1974 technical report took the transaction concept from semi-codified expression of practical knowledge to rigorous formalization, intersecting relational database theory in its early years. It stands today as a landmark publication in the history of automated data management.

Alastair Green, London, U.K.

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

BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/2601022 http://cacm.acm.org/blogs/blog-cacm

First Impressions, Unexpected Benefits

Daniel Reed shares his experiences with Google Glass, while Chris Stephenson considers the kinds of support chapters of the Computer Science Teachers Association provide their members.

Daniel Reed
Through A Google Glass, Darkly

http://cacm.acm.org/blogs/blog-cacm/173040-through-a-google-glass-darkly/fulltext
March 17, 2014

Whether de rigueur or merely an occupational hazard, a love of technological gadgets is commonplace in computing. We know the endorphin-fueled euphoria of an early adopter, one willing to tolerate the failings of immature and possibly transient technologies.

Oh, how I loved my RIM 870, which made me a two-thumbed typist and allowed me to send email on the DataTAC network, long before the word BlackBerry conjured any vision other than breakfast jam! I wore a Microsoft SenseCam, learning that email, coffee, and meetings defined my life.

My desk and closet are filled with "might have been" and "never were" technological artifacts. From a subcompact Poqet PC through an Apple Newton PDA to Ricochet network cards to a collection of bulky, CGA-based head-mounted displays (HMDs), I have played with my share of bright, shiny toys. Today, Google Glass shines oh, so bright, with technological promise and cultural uncertainty.

All of which reminds me of the first time I pulled out an early Motorola cellphone in an airport terminal, after an airline canceled one of my flights. This was in the early days of AMPS service, when mobile telephones were large and bulky and roaming service had yet to appear. As I called my office to rebook travel, a group of businesspeople gathered around me in wonder at this amazing and heretofore unknown technology. Years later, I was reflecting on the societal transformation wrought by mobile telephony as I listened to a middle manager obliviously pace an airport concourse while systematically firing the unsuspecting members of some company team via cellphone. *In ictu oculi, sic transit gloria mundi!*

I have been wearing Google Glass (hereafter, Glass) intermittently as both a technical assessment of utility and as a social study in human dynamics and expectations. What does the future hold? Prediction is very difficult, especially about the future. Attribute the quote to whomever you choose—Yogi Berra, Niels Bohr, or Mark Twain—its veracity remains.

Technical Experiences

My first impressions of Glass were generally positive, but I expected that; I have decades of Pavlovian conditioning to embrace technology prototypes with joyful abandon. Software configuration took most of an afternoon, as I connected my social networks to Glass and configured news feeds to balance timeliness and frequency, lest breaking news of the underwater basket-weaving championship interrupt my reflections on the dismal state of research funding. Not surprisingly, these networks were heavily Google-centric, though I found the inability to send images to an arbitrary email address somewhat limiting. Ok, Glass, take a picture ...

Because I am sufficiently myopic that escaping a broom closet is visually challenging, I wondered if it would be practical to wear Glass in front of my normal glasses, rather than fitting it to the custom Glass frames. Pleasantly, the answer is yes, if one carefully adjusts the prism. The screen is relatively bright and unobtrusive, and the head tilt adjustment angle simplifies screen activation and menu selection.

Is Glass a substitute for a smartphone? No, but it is not intended as such. Rather, it is a social and technical experiment in digital immersion and social commerce, from instant context sharing to walkabout video conferencing. In all of those things, it succeeds.

Social Challenges

Far more interesting than the technical features of Glass have been the social reactions it has engendered. When I am in a private setting, I always inform people that I will not take photos or record video using Glass without permission. I also invite people to wear Glass and experience it for a few minutes to allay fears. In public, I use Glass with great care, with as much deliberation and clarity I would use with a camera or smartphone.

The sometimes-visceral reactions to the public use of Glass are rooted in deep social concerns about privacy and cultural norms, themselves shaped by the broader worries about government surveillance, data breaches, and identity theft. Glass also occupies something of an uncanny valley of privacy, arousing primal fears that the sanctity of one's sensorium is being violated. The negative reactions have also been fueled by the annoying and sometimes foolish behavior of some Glass users. (See Google Glass "dos and don'ts," or how to avoid being a Glasshole, at http://bit.ly/1pMS15g.)

Each new technical change brings social and legal debates regarding acceptable use, debates made more pressing and contentious by the increasing rate of change. In the U.S., we are still defining the bounds of the First Amendment on the right to take photographs in a public place, despite over 150 years of photographic technology experience and an abundance of case law.

One of the creators of mobile telephony once remarked to me he was still surprised to see individuals speaking into thin air about intensely personal topics. That dystopian science fiction writer, Phillip K. Dick, favored the ancient line, "For now we see through a glass, darkly ..." I could wax on at length in a metaphorical and epistemological exegesis of this prescient juxtaposition of technology, privacy, and the nature of change, but a heads-up news feed on Glass has captured my attention.

Looking Forward

It is not surprising we are struggling to delineate the bounds of privacy and normative behavior in an age of ubiquitous digital devices. Some acceptable and normative issues of wearable technology will be decided by law, some by social consensus, and still others by concurrent technological evolution and generational acceptance. In the meantime, good sense should be applied liberally.

Despite all the uncertainties, I remain an optimist. Ok, Glass, send a message to the future ...

Chris Stephenson
Grant Proposal Time and the Unexpected Benefits of CSTA Chapters

http://cacm.acm.org/blogs/blog-cacm/172886-grant-proposal-time-and-the-unexpected-benefits-of-csta-chapters/fulltext
March 11, 2014

It is that time of year again. All across the country, computer science faculty members are preparing to submit their grant proposals. This means they are also sending requests to CSTA for involvement in their grants and for letters of support to accompany their grant proposals.

CSTA receives so many requests for grant project involvement and letters of support that we had to set up a protocol (http://bit.ly/1ecGl4J) to deal with them all. The protocol defines three different kinds of CSTA engagement:

- Principal or Co-Principal Investigator
- Paid Subcontractor
- Letter of support

Each of these is defined and, although each involves a different process for making the request, the CSTA Executive Committee uses the same criteria to evaluate every request. These are:

- The overall quality of the project (as described).
- The extent to which the project is consistent with CSTA's mission.
- The extent to which the project forwards CSTA's goals and objectives.
- The extent to which the project will benefit CSTA members.
- The extent to which the curriculum goals of the project reflect CSTA's curriculum priorities and documents.
- The extent to which the project may or may not be in conflict with proposals that CSTA is submitting to the same or similar grant programs or institutions.
- Institutional membership in CSTA.

These may seem like a lot of requirements for a letter of support, but this is the only way to ensure CSTA recommends only those projects worthy of the support of our 16,000 members, and that we use our resources, including our reputation, to support those institutions that, in turn, support CSTA and its members.

It is not surprising the most important factor in CSTA's involvement in national, regional, and local computer science education projects is the success of the CSTA regional chapters program. Thanks especially to the work of CSTA chapter liaison Fran Trees, there are now more than 53 CSTA chapters in the U.S. and Canada, and more are added each month.

Many of these chapters play a direct role in several very large grants from the National Science Foundation's CE21 grants program. They serve not just as peer-to-peer professional learning communities, but as centers for innovation, professional development, and advocacy. The chapters are also hothouses of CSTA's blossoming leadership programs and exemplars of mutually supportive relationships between K–12 educators and post-secondary faculty.

Over the years, it has become easier to understand why chapters prove so attractive to faculty members looking for grant partners. The chapters provide a direct link to teachers and students. They are a place where the ingenuity of research can meet the realities of classroom practice. The CSTA chapters provide an invaluable meeting of the minds for computer science educators of all levels.

Since their inception, the chapters have also been perceived as providing an invaluable link between K–12 computer science educators and post-secondary mentors. At SIGCSE, I learned CSTA's chapters are increasingly seen as an important resource for post-secondary faculty who are similarly in need of mentoring. As Dale Reed from the University of Illinois Chicago noted, "As computer science faculty in universities, we know a lot about computer science, but many of us have had absolutely no training in teaching. Being part of a CSTA chapter gives us access to people who can help us learn to be better teachers." As is true in the best cases, mentoring goes in both directions.

If you would like to be more involved in a CSTA chapter, contact me at c.stephenson@csta-hq.org and I will introduce you to a wonderful community of practice.

Daniel Reed is vice president for Research and Economic Development at the University of Iowa.
Chris Stephenson is executive director of the Computer Science Teachers Association.

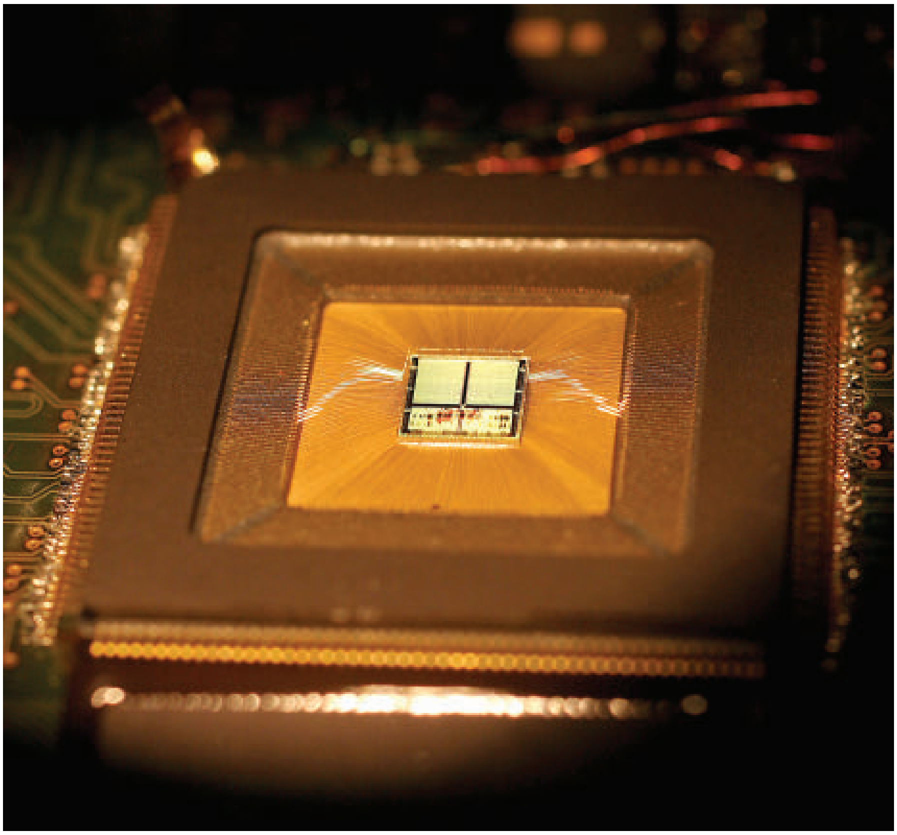
TECHNOLOGY
INSPIRATION
INNOVATION

N news

Science | DOI:10.1145/2601069 Don Monroe

Neuromorphic Computing Gets Ready for the (Really) Big Time

A technology inspired by biological principles but 'steamrolled for decades' prepares to take off as Moore's Law approaches its long-anticipated end.



"Spikey" is the first neuromorphic chip developed by the Electronic Vision(s) group at the University of Heidelberg, as part of the EU's Human Brain Project.

IMAGE COURTESY OF FACETS PROJECT. WWW.FACETS-PROJECT.ORG

As the long-predicted end of Moore's Law seems ever more imminent, researchers around the globe are seriously evaluating a profoundly different approach to large-scale computing inspired by biological principles. In the traditional von Neumann architecture, a powerful logic core (or several in parallel) operates sequentially on data fetched from memory. In contrast, "neuromorphic" computing distributes both computation and memory among an enormous number of relatively primitive "neurons," each communicating with hundreds or thousands of other neurons through "synapses." Ongoing projects are exploring this architecture at a vastly larger scale than ever before, rivaling mammalian nervous systems, and developing programming environments that take advantage of them. Still, the detailed implementation, such as the use of analog circuits, differs between the projects, and it may be several years before their relative merits can be assessed.

Researchers have long recognized the extraordinary energy stinginess of biological computing, most clearly in a visionary 1990 paper by the California Institute of Technology (Caltech)'s Carver Mead that established the term "neuromorphic." Yet industry's steady success in scaling traditional technology kept the pressure off.

"Neuromorphic computing people got steamrolled for decades because Moore's Law just kept getting better and better ... so they could just never catch up," says Todd Hylton of Brain Corporation, San Diego, CA, who in 2008 established the SyNAPSE (Systems of Neuromorphic Adaptive Plastic Scalable Electronics) program while he was working for the U.S. Defense Advanced Research Projects Agency (DARPA). "That's not the case anymore."

"It's a design trade-off: power versus complexity," says Gill Pratt, who now runs the SyNAPSE program. By employing a huge number of computational elements, each can be dedicated to a particular subtask. The inputs change only rarely, avoiding the energy overhead for moving charge and transporting data. "We believe this is the key reason" that biological circuits are so much more efficient, Pratt says. "You can only get away with it if size is free," which is effectively true with modern integrated circuits.

In addition to massive parallelism, the big neuromorphic projects explicitly track "spikes," the short pulses that carry information between biological neurons. When massively parallel artificial neural networks were studied starting in the 1980s, spikes were generally represented only by their average rate. Since the late 1990s, neuroscientists have found the detailed timing of the spikes in the brain conveys important information. None of the new chips reproduce the detailed spikes, only the time at which they occur, unsynchronized with any global system clock as in traditional chips. The guiding philosophy is not to reiterate or simulate the brain in complete detail, but to search for organizing principles that can be applied in practical devices.

The guiding philosophy is not to reiterate or simulate the brain in complete detail, but to search for organizing principles that can be applied in practical devices.

Spikes from hundreds of neurons are transmitted, via synapses, as inputs for another neuron, which combines the spike information to compute its own probability to fire off a spike to neurons to which it connects. The parameters that define that input and the computation for each neuron specify the chip architecture. Some teams perform this computation with analog circuitry, which reduces the power consumption at the cost of increased sensitivity to noise and device variability. Other researchers consider the power savings from the massively parallel architecture much more important, and have implemented the circuits digitally.

A key parameter is the synaptic strength, or "weight," which quantifies the contribution from each input neuron. The modification of synaptic weights, in response to local activity across a network, is a critical mechanism for learning. Narayan Srinivasa of HRL (formerly Hughes Research Laboratory) in Malibu, CA, who heads one of the two major collaborations funded under DARPA's SyNAPSE program, says replicating that learning, or plasticity, has been a central goal of his team. They designed chips that use the timing of the spikes to adjust the synaptic weights. The team found that a simulated chip discovers new features to look for in images without external coaching. "If the information changes, it will adapt," says Srinivasa. The latest challenge from DARPA, he says, is to train a version of their chip, with just 576 neurons and 73,000 synapses, to navigate a flying, hummingbird-scale, autonomous vehicle.

The other SyNAPSE collaboration, headed by Dharmendra Modha of IBM Research in Almaden, CA, has not yet focused on on-chip learning. Instead, the team is assembling a system of unprecedented scale, two billion cores with 100 trillion synapses—still smaller than a human brain, but comparable to that of a mouse. IBM implements neurons digitally, in part because the background leakage and variability of their state-of-the-art transistors is not well suited for analog circuits. In addition, doing everything digitally allows a perfect mapping between the circuit and simulations, Modha says. "Digital neurons are the key to success."

One requirement for exploiting these new chips is a new paradigm for programming them, Modha says. His team has developed a framework in which blocks of neurons are conceptually bound together to create "corelets" that perform a particular function, but whose inner workings are encapsulated in a fashion similar to object-oriented programming. This "end-to-end ecosystem" lets developers separately design corelets and combine them hierarchically into larger functional elements.

Modha says that even with digital neurons, "power is excellent" because of the massively parallel architecture. Yet in the Neurogrid project at Stanford University, Kwabena Boahen employs analog neurons, operating transistors in the ultra-low-power subthreshold mode championed by Carver Mead. These circuits often respond to electrical noise, he says, but researchers are looking for organizing principles so "reliability arises at the system level, not from the individual components." Like Modha, Boahen sees software support as critical to using the new systems, and is working with Chris Eliasmith at the University of Waterloo, Canada, to develop a "neural compiler" that lets developers move their proven algorithms to the new architecture.

Many neuromorphic projects mix two distinct goals: exploring biology and improving technology. However, "understanding the brain itself is different from building a better handset," says Tony Lewis of Qualcomm, in San Diego, CA. "I think you have to pick one or the other." His company has picked applications, announcing plans for a "neural core processor" called zeroth. The chip would work alongside traditional digital chips "to bring the sort of intelligence that people usually associate with the cloud down to the handset;" for example, image analysis and voice recognition. Lewis says the company does not intend to build these applications, just to make them possible. "We're enablers," he says, "not the guys who are going to write the final applications." Qualcomm announced its chip plans much earlier than usual in order to get feedback on the software tool chain that will let programmers take advantage of the new architecture when it is available.

Qualcomm also provides venture funding to collocated but independent Brain Corporation, founded by neuroscientist Eugene Izhikevich. Hylton, who moved to the company from DARPA in 2012, says it is eager both to understand brains and to develop autonomous robots, "but we're looking to solve it with a variety of approaches," not exclusively neuromorphic computing. In the end, some algorithms might be implemented using traditional hardware, as happened with neural networks. Hylton says that, although adaptation to novel environments is important, it will be important to share the learning to give each device a baseline capability. "All the robots don't have to go to school."

The European Union is also pursuing neuromorphic computing in a big way, as one piece of the billion-euro Human Brain Project (HBP). However, "the European sense is that this whole area is still held back because the basic principles of operation in the brain are still a mystery, and if only we could crack that, then we'd make giant leaps forward," says Steve Furber of the University of Manchester in the U.K. "On the U.S. side…there's a sense that 'we can build little cognitive units, let's go see what applications we can find.'" For a decade, Furber has been running the SpiNNaker (Spiking Neural Network Architecture) project, which combines ARM cores in a massively parallel network. HBP funding will move the project to a much larger scale, a million cores.

SpiNNaker's digital architecture is complementary to the other HBP neuromorphic project, headed by Karlheinz Meier at the University of Heidelberg, which uses mixed-signal neurons previously explored in the BrainScaleS (Brain-inspired multiScale computation in neuromorphic hybrid Systems) project. To reduce the need to drive off-chip interconnections, this project uses wafer-scale integration of 200,000 neurons and 50 million synapses with a relatively-cheap 180 nm process on an 8-inch-diameter wafer. "We decided to be faster than biology: not milliseconds, but tens of nanoseconds," Meier says, which lets researchers study slow processes like learning. "In our system, you need 10 seconds [to simulate] a day." With the support of HBP, the team is building a 20-wafer system that will have four million neurons and one billion synapses, to be completed in two years.

Even the most ambitious of the new neuromorphic projects are idealized abstractions of biological brains; they assume that only the timing of spikes is important, they replace the three-dimensional "wetware" geometry with two-dimensional networks of fast wires carrying multiplexed events, and the feedback paths that help fine-tune brain performance are largely missing. Still, "it's a very exciting time," says DARPA's Pratt. The scale of these projects "allows you to show things you don't see when they're small."

Will spike-based neuromorphic systems be more successful than the artificial neural networks of a quarter century ago? Actually, "neural networks were not a failure," says Meier. "The algorithms were incorporated into deep learning," like that employed by Google for facial recognition.

Yet neuromorphic systems hold the promise of going beyond algorithms to specialized hardware that cashes in the enormous complexity of today's integrated circuits to do large computations with low-power mobile devices. **C**

Further Reading

"A New Era of Computing Requires a New Way to Program Computers," Dharmendra S. Modha, http://bit.ly/1i2rxqh

"Low-Power Chips to Model a Billion Neurons," Steve Furber, *IEEE Spectrum*, August 2012, http://bit.ly/1fuotGv

"Cognitive Computing," Dharmendra S. Modha, Rajagopal Ananthanarayanan, Steven K. Esser, Anthony Ndirango, Anthony J. Sherbondy, Raghavendra Singh, *CACM*, August 2011, http://bit.ly/1i2rtqz

"A Computer that Works like the Human Brain," a TEDx talk with bioengineer Kwabena Boahen, http://bit.ly/1gOY0UM

Don Monroe is a science and technology writer based in Murray Hill, NJ.

ACM Member News

MAAREK LEADS YAHOO! SEARCH ENGINE TEAMS IN ISRAEL, INDIA

New ACM Fellow Yoelle Maarek is vice president of research and the head of Yahoo Labs in Haifa, Israel, where the search engine expert leads a team of 60 scientists and research engineers co-located in Israel and India in utilizing big data to solve complex search problems and develop new products.

Her team's dual mission: positively impact Yahoo systems and the academic research community. The French-raised Maarek is well suited to these tasks; she earned her Ph.D. in computer science from Technion – Israeli Institute of Technology, and did stints at IBM Research in the U.S. and Israel concentrating on search technologies. In 2006, she joined Google to open its first Haifa-based Engineering Center; she moved to Yahoo in 2009.

A typical day for Maarek is "a mixture of technical contributions, design reviews, writing papers, developing code, testing and debunking theories and technologies, and talking to partners and colleagues worldwide."

Maarek thrives on research challenges, "especially those involving huge amounts of data and hundreds of millions of users." That means conducting data-driven experiments to demonstrate working principles for new search engine functionality and products. She tries to maintain equilibrium amid the triangle of conflicting usage data factors: Rightsizing big data + Personalization + Data Privacy.

"Our goal is to invent something that no one has done yet," she said.

Living in the world's Fertile Crescent is advantageous and invigorating. "I have the best of all worlds. I love my job. I can communicate with Yahoo developers in California and India, take my team to the beach to work, and go hiking with my family," Maarek said.

—Laura DiDio

Technology | DOI:10.1145/2601075

Neil Savage

Time for a Change

4D printing combines the dimension of time with the hope of building objects with new capabilities.

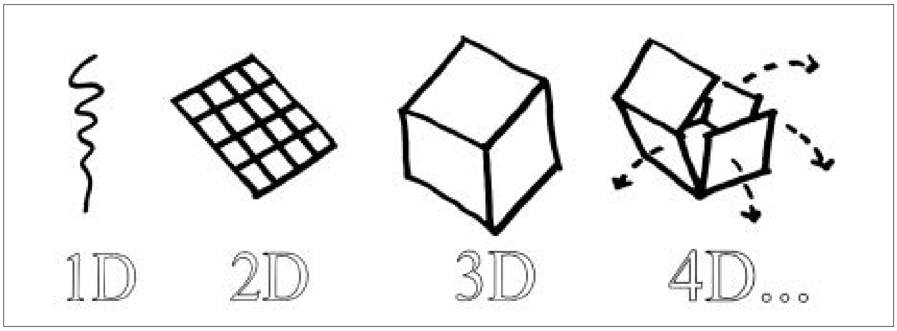
IMAGINE A PAIR of running shoes that grow spikes on their soles to gain traction when it starts to rain in the middle of a run, or camouflage material that changes color as the light changes. Those are just some of the applications that might become possible with a new technique known as 4D printing.

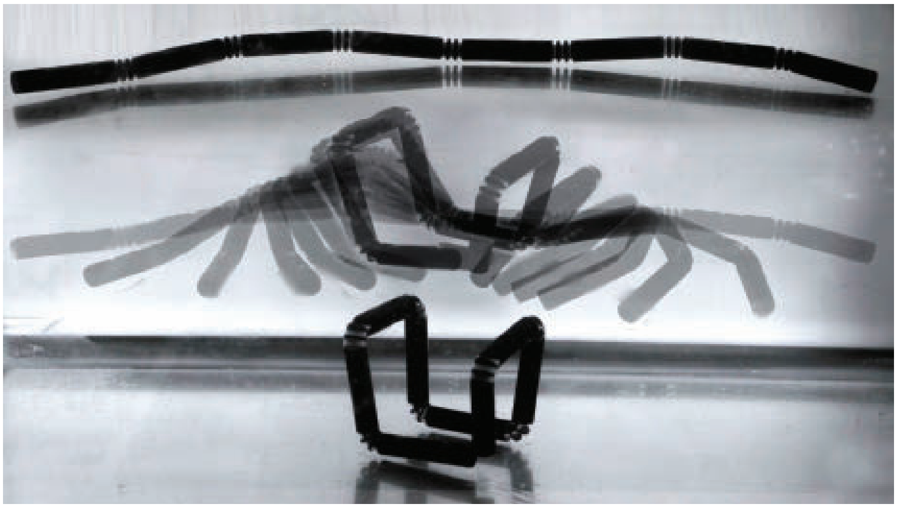
The concept starts with additive manufacturing (3D printing), in which a machine builds a three-dimensional object by depositing successive layers of a material—polymers, resins, metals, ceramics—in almost any shape a designer can imagine. 3D printing has been garnering headlines and gaining a foothold in the world of manufacturing in recent years. The industry analysis firm Wohlers Associates says the worldwide market for 3D printing products and services reached $2.2 billion in 2012. Now some researchers are taking the next step by adding a fourth dimension—time.

"We want to be able to print objects, products, materials that transform over time," says Skylar Tibbits, a research scientist who runs MIT's Self-Assembly Lab in Cambridge, MA. "Why can't we print dynamic things, or responsive things?"

The idea is to build an object out of multiple materials using a 3D printer. The materials would have different properties—their stiffness, say, or their sensitivity to temperature or moisture. For instance, one material might shrink a lot when heated, while another would not. By placing those different materials in the right configuration, a designer could determine what shape the object would take when heated.

The idea of materials that change their shape in response to some stimulus is not unique to 4D printing; other researchers have worked on "programmable matter," such as thin sheets of solar cells that can unfold in response to a temperature change. Combining that idea with 3D printing, though, marries the usefulness of active materials to the new approaches to manufacturing provided by the printers.





Above, a basic depiction of how a 3D printed object that changes shape or self-assembles over time becomes a 4D printed object. Below, one of Skylar Tibbits' 4D objects; this 3D printed "self-folding straw" starts out as a cable-like structure, but it assembles itself into a pre-programmed shape when it encounters water.

For instance, being able to print something in one form and having it open up into another makes it possible to print objects that are larger than the 3D printers. Also, printing something in a compact form could involve less wasted material, and take less time, than printing it in its final shape. If an active object can be printed in a single piece, without having to screw in or glue on additional components, it should be cheaper and quicker to build and have fewer possible ways to break. "The more components you have, the more failure-prone your system is," Tibbits explains.

In one demonstration, Tibbits printed a black polymer into a flat disk. He then added to one side of the disk four concentric rings of a silver-colored polymer. Halfway between each pair of rings, but on the opposite side of the disk, he placed more rings of the same silvery polymer. When he immersed the disk in water, it bent along the rings, upward where the rings were on top and downward where they were on the bottom, in accordion-like pleats, forming a ridged, saddle-shaped object.

He also made a series of joined squares, with ridges where they intersect; when he wets that, it folds along the ridges into a cube.

IMAGES COURTESY OF SELF-ASSEMBLY LAB/MIT

Most of Tibbits' work so far has focused on demonstrating that this process actually works, and figuring out ways in which to design responsive objects. He is not working on building any sort of commercial products, but says some may emerge in just a few years. One early application, he projects, will probably be sports apparel: a piece of clothing might react to an athlete's body heat and open up pores to cool the athlete down, or close up if it sensed rain. Another use, he says, might be in infrastructure, such as water pipes that expand or contract to alter water pressure or flow rate.

While those kinds of applications would require that changes be repeatable and reversible, in other uses it would be enough to have a shape that changes only once. One notion that appeals to Carlos Olguin, head of the Bio/Nano/Programmable Matter Group at Autodesk Research in San Francisco, CA, is to create self-assembling furniture. A customer might purchase, say, a chair from a store and carry it home in a small, compact form. Once home, the customer would apply whatever stimulus the material was designed to respond to, and the clump would unfold into a chair that would then maintain its shape.

"4D printing in many ways is just another form of programming matter," says Olguin, who collaborates with Tibbits. Olguin is working on developing the computer-aided design tools that will let engineers figure out where to place the material to get the function they are seeking. His work is part of Autodesk's Project Cyborg, which is developing a CAD platform that uses computational resources in the cloud to provide modeling, simulation, and design optimization for material at different size scales.

Olguin's hope is that a user working with traditional CAD software could design a desired object—a customized version of that chair, for instance—and that the computer could then apply a transformation algorithm to come up with the shape that, once printed, could become the chair. To do that, the system would need plenty of information about the characteristics of the materials the printer would use, and would have to arrange the materials in a way that would lead to the outcome the user intended. That is still a very tough problem, Olguin says. "It's not just about software coding. It's about getting materials to behave in the way that you want."

> "It's not just about software coding. It's about getting materials to behave in the way that you want."

Tibbits would also like to be able to give a computer a description of the product he wanted—not only its shape, but its abilities—and have the machine figure out just which materials to use and where to place them. When he started this research a couple of years ago, most of what he did was empirical: building things and seeing how they worked. Since then, he has been building up more data about the properties of the materials he works with, which allows him to do more simulation before actually building a device. While he is investigating the 4D-friendly properties of existing polymers, he hopes materials scientists will also develop new materials with characteristics a 4D printer might be able to utilize.

All of Tibbits' work so far has used water as the catalyst to make an object change shape. Jerry Qi, who studies the mechanics of materials at Georgia Institute of Technology in Atlanta, focuses on the fact that many polymers exhibit shape memory behavior with temperature changes. Heated above a certain point, the polymers become pliable enough to be bent into a shape, which it holds when it has cooled; heat it to the same temperature again, and it reverts to its original shape. The temperature at which the transition happens is different for each polymer.

Qi uses a 3D printer that heats and deposits polymer fibers. By controlling the diameter and orientation of the fiber, and how close to the surface he places it within another polymer, he can control how the fiber bends. "You have a

Special Interest Groups

Newest ACM SIG Focuses on Logic

ACM takes great pleasure in announcing the formation of its newest special interest group: SIGLOG, the Special Interest Group on Logic and Computation.

Among other activities, SIGLOG will organize conferences and workshops, and serve as an advocate for the importance of logic in the undergraduate computer science curriculum. The group also plans to establish awards to recognize outstanding contributions made by members of the Logic and Computation community.

The new SIG's flagship conference will be the ACM-IEEE Symposium on Logic in Computer Science, which will take place this year July 14–18 in Vienna, Austria. In addition, July's Federated Logic Conferences in Vienna (part of the Vienna Summer of Logic) will feature a SIGLOG launch event.

SIGLOG plans to publish a quarterly electronic newsletter featuring community news, technical columns, members' feedback, conference reports, book reviews, and other items of interest to the community.

For membership information, please contact ACM Member Services at (212) 626-0500 or by email at acmhelp@acm.org. For all other information, please contact SIGLOG chair Prakash Panangaden at prakash@cs.mcgill.ca.

lot of freedom," he says. "You can control what kind of deformation you get."

Depending on the material used and the design of the object, the temperature difference needed to induce a change can be fairly small. One potential use Qi sees is for a medical device such as a stent, which can be built in compact form for easy placement inside an artery; once in place, the body's own heat would make it expand to fit.

So far, these researchers are working exclusively with polymers. Though it is possible to get some shape-changing behavior with a single material and clever design, it works better with multiple materials. The only 3D printers currently on the market that are capable of printing multiple materials come from Stratasys, and those print using only different types of polymer. Many researchers are working on ways to print more disparate materials together so that they could, for example, deposit metal wires inside ceramics, or build plastic parts with embedded circuitry. Because the processes for printing with different materials vary widely—the temperatures that melt metal to form parts also vaporize plastic, and inkjet printers that work nicely with one type of ink quickly clog with another—it could be a few years before more complex material designs are realized.

Being able to incorporate more capabilities, including electronics, in a 3D printed object could lead to devices with even greater abilities. "You could add sensing, maybe actuating capabilities, into your structure," Qi says. One application might be used as the skin of an airplane or the heat-shielding tiles on a space shuttle; the material might be able to sense damage and alter its shape to self-heal, forming a temporary patch to protect the craft until it can be properly repaired.

Olguin argues that another area of research—3D bioprinting—might also be considered a form of 4D printing.

Olguin argues that another area of research—3D bioprinting—might also be considered a form of 4D printing. Researchers are trying to find ways to incorporate living cells into 3D printing, depositing them along with a soft matrix to form artificial organs or biorobots. In that case, Olguin says, the cells that grow and change over time bring in the fourth dimension.

The U.S. military believes 4D printing has potential. The U.S. Army Research Office last fall gave an $855,000 grant to a trio of researchers from the University of Pittsburgh, Harvard, and the University of Illinois to develop materials that would be useful to soldiers. Lead researcher Anne Balazs, a chemical engineer at Pittsburgh, says the group is concentrating on manipulating materials at a microscopic level to achieve desired properties, which might involve active camouflage or protection from shrapnel.

Though the field of 4D printing is still in its infancy, the researchers say it is likely to be only a few years before 3D printed objects that change over time start making their way into everyday use. "The momentum's there," Qi says. "This field is developing really fast."

Further Reading

Ge, Q., Qi, H.J., Dunn, M.
Active Materials by Four-Dimensional Printing, *Appl. Phys. Lett 103*, 2013.

Tibbits, S.
4D Printing: Multi-Material Shape Change, *Arch. Design 84* (1) 116-121 (2014)

Goldstein, S.C., Campbell, J.D., Mowry, T.C.
Programmable Matter, *Computer 28* (6), 99-101 (2005)

Change Generation: Skylar Tibbits
http://vimeo.com/70660109

Neil Savage is a science and technology writer based in Lowell, MA.

Milestones

Computer Science Awards, Appointments

DUAL HONORS FOR PURDUE'S SPAFFORD

Purdue University computer science professor Eugene H. Spafford, a recognized leader in the information security field, recently was inducted into the National Security Hall of Fame.

Spafford, who also is executive director of Purdue's Center for Education and Research in Information Assurance and Security, also was selected to receive the Harold F. Tipton Lifetime Achievement Award.

Spafford, a Purdue faculty member since 1987, was one of the first Morrill Award recipients at Purdue University. He is chair of ACM's U.S. Public Policy Council (USACM), a Fellow of the ACM, a Fellow of the American Association for the Advancement of Science, a Fellow of the Institute for Electrical and Electronics Engineers, and charter recipient of the Computer Society's Golden Core award.

VARGHESE RECEIVES IEEE COMPUTERS AND COMMUNICATIONS AWARD

George Varghese, a principal researcher at Microsoft Research Redmond, recently was named the 2014 recipient of IEEE's Koji Kobayashi Computers and Communications Award, "for contributions to the field of network algorithmics and its applications to high-speed packet networks."

The award is presented for outstanding contributions to the integration of computers and communications.

Previous recipients include ACM president Vint Cerf, ACM Fellow Robert Kahn, and ACM Software System Award recipient Timothy Berners-Lee.

ROYAL SOCIETY OF CANADA NAMES AHO 2013 FELLOW

The Royal Society of Canada (RSC), a leading Canadian society of distinguished scholars, artists, and scientists, named computer scientist Alfred Vaino Aho as one of its 2013 Fellows.

The RSC said Aho, who holds the Lawrence Gussman Chair of Computer Science at Columbia University, "has made fundamental contributions to string searching, databases, formal languages, programming languages, compilation, and pattern matching. His research is used daily in millions of computers worldwide."

Aho is a Fellow of the American Academy of Arts and Sciences, the American Association for the Advancement of Science, ACM, Bell Labs, and IEEE.

Society | DOI:10.1145/2601074

Visualizations Make Big Data Meaningful

New techniques are designed to translate "invisible numbers" into visible images.

Until recently, a good spreadsheet or perhaps a pie chart might have been sufficient to get a firm grip on a dataset. However, making sense of big data requires more, and with our increasing inundation with data comes new and creative opportunities to build unique interfaces.

That is why, powered by new technologies such as touchscreens and giant LCD monitors, so-called "data artists" are experimenting with ways to make huge amounts of data comprehensible and accessible to a broader range of people, and then displaying their data visualizations inside venues like the corporate facilities of companies like Microsoft and Google.

Whether people are familiar with the term "data visualization" or not, they are certainly aware of some of the more popular implementations of it. For instance, the Facebook Timeline is a visualization tool designed by data artist Nicholas Felton as a way of manipulating and organizing the information in Facebook's database.

"Those who create data visualization are providing the beautiful, easy-to-use interfaces that lie on top of massive amounts of number crunching from a growing array of sources," observes Justin Langseth, CEO of Washington, D.C.-based startup Zoomdata.

Indeed, like all art, data visualizations come in many shapes and flavors—from the inventive software-based work of Jer Thorp to the big-screen animated films from Pixar Animation Studios; from the business solutions of Zoomdata to the documentary "Clouds" by James George on the emerging field of data visualization (filmed using Microsoft Kinect's Depth camera paired with a digital SLR video), which premiered recently at the Sundance Film Festival.

An image from "Clouds," a "computational documentary" by James George in collaboration with Jonathan Minard, which depicts people discussing creative use of code.

IMAGE BY JAMES GEORGE AND JONATHAN MINARD

In fact, George's work in video and photography epitomizes the relationship between art and computer science. Besides being an independent artist, he is also the first artist-in-residence at Microsoft Research's Studio 99 Gallery, and is a lecturer on computational processes in video art at New York University's Interactive Telecommunications Program.

One of George's experiments in computational photography began with a story he heard on the radio about a surveillance system that was to be installed to combat crime in the New York City subway system. Cameras were to be mounted in all subway stations and, instead of having the monitors watched by people, AI algorithms would be used to identify threats.

"The system failed miserably because there were too many passengers, too much data being generated," George recalls. "So the system was switched off, but the cameras are still there today, attached to computers that aren't turned on."

George and his team had been experimenting with the Microsoft Kinect camera, which sees the world in three dimensions, transforming it into a dataset that can be manipulated in post-production.

"We took the camera into the subway system and began capturing image data with the intention of re-creating the way the security system might have seen the world if it had worked," he explains. "We called it 'DepthEditorDebug,' after the name of the app we used to make it, and what it did was illustrate that, as in any art form, there is a level of interpretation depending on what you leave in, what you leave out, or what you fix—known as 'cleaning the data.' So data visualization is a very expressive medium in the same way that painting can be expressive. The role of the artists is to apply their insight to the data and then choose what to show and how to show it."

A co-founder of The Office For Creative Research, an R&D group for hire, Jer Thorp says his team is hired by corporations and museums to approach novel problems concerning data and to engage with that data in ways that probably have not been done before. His two biggest clients are currently Microsoft and the Musum of Modern Art (MOMA) in New York City (where he and his colleagues collectively serve as artists in residence).

"Regardless of whom we work for, everyone is being overwhelmed by data, and the promise of simplification becomes really attractive," Thorp says. "Being able to see a single graphic that represents a complicated thing makes peoples' cognitive load a little easier."

The ability to lighten that load, he says, comes from three advances in technology: the fact that storage of data has become almost free; the cloud and GPU-driven calculations that have opened computing to the masses; and the availability of software tools like Apache Hadoop that have simplified the processing of large-scale datasets.

> **"Being able to see a single graphic that represents a complicated thing makes people's cognitive load a little easier."**

Take, for example, the project Thorp completed last year for the Vancouver Art Gallery called "Grand Hotel." Starting with a database of about 700,000 hotels—not every hotel in the world, but pretty close—he and his team wondered what would be the most engaging way to visualize all that information; the hotels' ratings, their locations, photos, and so on.

"We could have plotted points on a map or built graphs around their room rates," he says, "but instead, we imagined what it would look like if characters from famous literature were to make their travels today; where would they stay, what would they think of the hotels. It was a way to display the data we had, but in an entertaining fashion."

They started with characters from the book *Lolita* and had them travel down the eastern coast of the U.S., stopping at rundown hotels as they do in the novel, with information about those hotels projected on a 16-by-30-foot screen.

"Then we got more abstract and did the same with Ulysses and his army, plotting all the points where, theoretically, he would have stayed at hotels," Thorp says. Then they did the same with *Lawrence Of Arabia*, Jack Kerouac's *On The Road*, and other traveling characters of literature.

"The data visualization is in some ways a data narrative," says Thorp. "It explains, but it also relates. It intrigues. It compels. It allows us to unpack that dataset in a way which I feel is a lot more human than it might have been if it were just, say, a chart or spreadsheet."

Humanizing the use of leading-edge technology is also the task of Tony DeRose, who heads up the Research Group at Pixar, the animation

International

Panel Mulls Outlook for European CS Research

A special open session during a recent ACM Europe Council meeting at the Royal Society in London included a panel discussion moderated by former ACM president Dame Wendy Hall on "The Future of Computer Science Research in Europe."

The panel, made up of representatives of government, industry, and academia, addressed weaknesses in Horizon 2020, a program in which the European Union has pledged to invest nearly 80 billion euros (about $110 billion) through 2020 to ensure Europe's global competitiveness in research and innovation. As noted in the Information Society Technologies Advisory Group (ISTAG) Working Group on Software Technologies report "Software Technologies: The Missing Key Enabling Technology," the Horizon 2020 plan identified numerous "key enabling technologies" but failed to include software development as one of them.

Chris L. Hankin of Imperial College, who is on the European Commission Directorate General for Communications Networks, Content & Technology (DG Connect) Advisory Forum (CAF), addressed the Digital Agenda for Europe, the EU's strategy to help digital technologies deliver sustainable economic growth to "reboot Europe's economy." As the first of seven flagship initiatives under Europe 2020, the EU's strategy to deliver sustainable and inclusive growth, Hankin said the Digital Agenda aimed to double ICT (information and communication technologies) R&D spending from 5.5 billion euros to 11 billion euros by 2020.

He added, "It's really the responsibility of groups like ACM Europe that that kind of investment is properly directed and achieves the kind of impact that the Digital Agenda for Europe foresaw."

Robert Madelin, EC DG Connect's director general, said, "I think we're going in the right direction, but the 'right direction' has to be more explicitly open to cooperation upstream with different interests, and it has to be more engaged with ordinary people and issues."

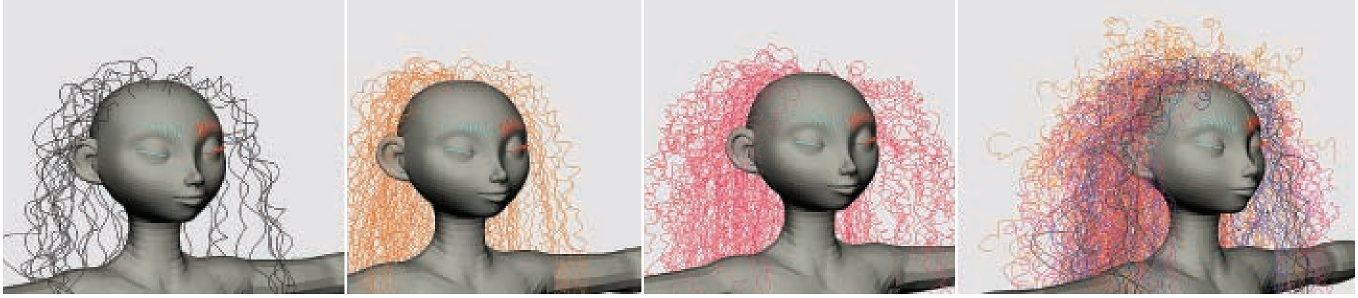
Paola Inverardi, first female rector of the University of L'Aquila, Italy, and Italy's representative to the EC for ICT, says Horizon 2020 "to some extent reduces ICT and computer science to an ancillary focus."

Last year's High-Level Vision 2030 report by industry association ARTEMIS projected jobs in software and services for Europe will reach about 8.9 million, Inverardi said, compared to less than a million working on semiconductors and other hardware. "We have a mass of people working on the soft part of ICT, and I think that we need to take care of their future."

Microsoft Research Connections vice president Tony Hey stressed the importance of teaching computer science to "stimulate young people into the potential of science." However, he added, " it seems absolutely incredible if you don't have a serious computer science research agenda in Horizon 2020. That's where your future is going to be."

Andreas Reuter, managing director of the Heidelberg Institute for Theoretical Studies, said, "I think funding agencies should not primarily try to fund directions of research ... they should rather address the environmental conditions, political, legal, economical infrastructure that will foster leading-edge research and will support the take-up of research results by industry and society."

—Lawrence M. Fisher



The challenges of visualizing lifelike hair in the animated fantasy *Brave*.

IMAGE COURTESY OF DISNEY/PIXAR, FROM PIXAR.WIKIA.COM

studio known for its big-screen features. "Our group tries to chip away at the big, hard, technical challenges coming down the pike," he says, "especially the ones where the probability of failure is high."

One such challenge that sounds simple was creating lifelike hair on Merida, the main character in *Brave*, the 2012 animated fantasy about a princess in the Scottish Highlands. To create her mass of long, bouncy red hair that had to fall on her shoulders in a believable fashion, DeRose's team needed to create a mathematical model to describe how hair should move, and then devise a simulation system that would use that model to create the on-screen motion.

"That subtle motion is something an animator isn't going to want to go in and control every little detail by hand," DeRose explains.

One of the interesting applied math/computer science questions was what gives real hair its volume. It turns out that approximately 100,000 hairs rest on each other, colliding with the hairs around them as a head moves. "The potential number of collisions is enormous—about 100,000 squared potential collisions," DeRose says. "So we needed to develop algorithms that would quickly determine which subset of hairs to check for collisions. If we had to do all of that by hand, we'd still be waiting for the images to complete themselves."

What has enabled such data manipulation, says DeRose, is the computing power "that has gone through the roof in the last 10 years. GPUs being developed today are opening up new levels of complexity that we'll be able to bring to the screen. As John Lasseter, our chief creative officer, has said, 'the artistic vision inspires new technical development ... and new technical development inspires new art.' And so we get this nice, positive feedback cycle between the two disciplines, technology and art."

Meanwhile, in the world of commerce, companies like Zoomdata are using data artists to design interfaces strictly for business purposes. "The business world is pretty adept at creating pie charts and bar charts," says Zoomdata CEO Langseth, "but it thirsts for more interesting, more intuitive interactive visualizations of data. That's the space we're in: coming up with novel, dynamic ways of looking at the data, as opposed to staring at tables and columns."

For instance, one Zoomdata "visualization" is a cascading chord chart that displays connections from one place to another, perhaps the flow of traffic on a Web site from one area of the site to another.

"The most competitive retailers, for example, are using advanced analytics to track customers and to target advertisements very specifically to the information they have on those customers," says Langseth. "And that involves new sources of data and new ways of looking at and exploring that data.

"Since Apple reinvented the phone, many Silicon Valley startups have three co-founders: a technologist, a business person, and an artist who has recently become critical to almost everything people do with computers," he says. "And data is no exception. It's not something that's received much artistic attention over the years, but now companies like ours are starting to focus on that."

Indeed, Google is sponsoring a global competition to find an up-and-coming software developer who pushes the boundaries of art using data and code, which Google calls "DevArt." The winner of the competition will receive a $41,000 prize, as well as Google Developer assistance and curating, and production support from the Barbican, Europe's largest multi-arts venue in London, to help transform their concept into a digital art installation.

"I sometimes hear phrases like 'artists create, developers code,' but nothing could be further from the truth," says Google developer advocate Paul Kinlan in a recent blog. "We are all a creative bunch with a passion for exploring and creating amazing works that push the boundaries of what we believe is possible with modern computing technology. Sometimes, we just need some inspiration and an outlet." C

Further Reading

"DepthEditorDebug," a video and photographs posted February, 2011 by James George and Alexander Porter, at http://jamesgeorge.org/works/deptheditordebug.html

"Clouds," a video documentary posted January, 2014 by James George and Jonathan Minard, at http://jamesgeorge.org/works/clouds.html

"Jer Thorp: The Weight Of Data," a video posted February, 2012 by TEDx, at http://www.youtube.com/watch?v=Q9wcvFkWpsM

"Math In The Movies," an article posted by the Mathematical Association of America, at http://www.maa.org/meetings/calendar-events/math-in-the-movies

"Pixar And Math – Disney Pixar's *Brave* – Wonder Moss," a video posted November 2012 by Inabottle, at http://www.youtube.com/watch?v=EnaA9ZRPXiE

"Data As Paint, And The Rise Of The Data Artist," a blog posted March, 2013 by Justin Langseth, at http://datametaphors.wordpress.com/

"DevArt: Your Code Belongs In An Art Gallery," a blog posted February, 2014 by Paul Kinlan, at http://googledevelopers.blogspot.com/2014/02/devart-your-code-belongs-in-art-gallery.html

Milestones | DOI:10.1145/2601076 Neil Savage

General Agreement

Leslie Lamport contributed to the theory and practice of building distributed computing systems that work as intended.

"WHAT COMES FIRST?" is critically important when computer processors perform operations that rely on each other's outputs. Leslie Lamport realized this when reading a 1975 paper by Paul Johnson and Robert Thomas proposing timestamps to track the sequence of events in a distributed algorithm.

"I immediately related that to special relativity," Einstein's theory of spacetime, says Lamport, principal researcher at Microsoft Research and recipient of the 2013 ACM A.M. Turing Award. "The notion of 'at the same time' is relative. It's not an invariant notion. But what is invariant is the notion of causality." Lamport will receive the $250,000 Turing prize for his work in bringing order to the chaos of distributed computing systems that rely on communication between many autonomous computers.

According to Einstein, two events can only be causally related if light from the first event can reach the other before the second event happens. Lamport says the same notion can be applied to messages in a computer system; the output of one processor has to reach a second processor before the second one performs an operation, if the first is to influence the second. To make that happen, you need to define "before."

That is what Lamport did with the notion of logical clocks, which count a sequence of clicks and assign a clock value to each event. For one event to occur before another, it must have a lower clock value. Once the sequence of events is determined, it becomes possible to build an arbitrary state machine. "Then you can implement any type of system you want," Lamport says.

Logical clocks work fine when nothing fails, Lamport says; the problem is how to get such a system to work in the presence of faults. From 1977 to 1985, Lamport worked for research institute SRI International, which had been contracted by the U.S. National Aeronautics and Space Administration (NASA) to build a prototype of a highly reliable system for controlling aircraft. It was assumed a three-processor system could tolerate one faulty processor; if two processors took input from a sensor and calculated similar results, but a third came up with a wildly different value, the two similar results could be considered correct and the third discarded.

The problem, laid out in a paper by Lamport and his SRI colleagues Robert Shostak and Marshall Pease, is the faulty processor might send different values to each of the other two; both processor A and processor B might think processor C was in agreement with it and the other processor was faulty. The group realized another processor was needed; in fact, to tolerate a given number n of faults required three times as many processors as that number plus one more, or 3n + 1. The paper won the 2005 Edsger W. Dijkstra Prize in Distributed Computing.

Lamport credits his colleagues with that solution; he says his contribution was a different algorithm using digital signatures so the processors could identify which one gave inconsistent results, so the problem could be solved with 2n + 1 processors. At the time, cryptographic signatures were too expensive to implement, so his solution was not used.

He takes credit for getting the paper written. "I wrote the first draft, that Shostak thought was so terrible he completely rewrote it," Lamport says.

"The notion of 'at the same time' is relative. It's not an invariant notion. But what is invariant is the notion of causality."

Shostak says Lamport's most important contribution was naming the problem when, in another paper, he described the processors as Byzantine generals surrounding an enemy encampment, deciding whether to attack. The faulty processor was described as a traitorous general, telling one colleague they should attack and the other they should not. The solution is now called Byzantine agreement.

"That made a big difference in terms of the propagation of our work," says Shostak. Lamport "has a knack, I think, for popularizing results in computing science, in addition to being somebody who has a large corpus of original work."

Lamport says that approach failed with Paxos, an algorithm for a fault-tolerant state machine handling non-Byzantine failures. He illustrated it with the story of a parliament on the Greek island Paxos reaching agreement. Though the algorithm is useful, Lamport says, the story did not help promote it.

Cornell University computer scientist Fred Schneider says Lamport's strength lies in how he takes a "first principles" approach to computer science problems, laying bare underlying assumptions others might not have noticed. "He doesn't accept common dogma or common descriptions of problems," Schneider says.

Lamport replaced the older definition of correctness, regarding whether a program terminated correctly, with the notions of safety—that nothing bad happens—and liveness—that something good happens. He also developed LaTeX, a document preparation system standard for publishing documents in some scientific fields. "Every time he put his finger down, he picked up a gem," Schneider says.

Lamport's advice for those who want to become systems designers: learn to think better. "The way you learn to think better is to think more mathematically," he says, "so they should get themselves a good mathematical education." C

Neil Savage is a science and technology writer based in Lowell, MA.

PHOTOGRAPH BY RICHARD MORGENSTEIN

DOI:10.1145/2602321

Ross Anderson and Steven J. Murdoch

Inside Risks
EMV: Why Payment Systems Fail

What lessons might we learn from the chip cards used for payments in Europe, now that the U.S. is adopting them too?

U.S. CREDIT CARD companies and banks are beginning to distribute new credit cards with an embedded chip as well as the magnetic strip that has been in use since the 1970s. Named for its promoters Europay, MasterCard, and Visa, the EMV system augments the old magnetic strip cards with a chip that can authenticate a transaction using cryptography—a so-called "smartcard." EMV was deployed in the U.K. from 2003 to 2006 and in other European countries shortly afterward; it is now being rolled out from India to Canada. The idea was to cut fraud drastically, but real-world experiences turned out to be somewhat more difficult than theory. As shown in Figure 1, fraud in the U.K. went up, then down, and is now heading upward again.

The idea behind EMV is simple enough: The card is authenticated by a chip that is much more difficult to forge than the magnetic strip. The cardholder may be identified by a signature as before, or by a PIN; the chip has the ability to verify the PIN locally. Banks in the U.K. decided to use PIN verification wherever possible, so the system there is branded "chip and PIN"; in Singapore, it is "chip and signature" as banks decided to continue using signatures at the point of sale. The U.S. scheme is a mixture, with some banks issuing chip-and-PIN cards and others going down the signature route. We may therefore be about to see a large natural experiment as to whether it is better to authenticate transactions with a signature or a PIN.

The key question will be, "better for whom?" The European experience suggests this will not be a straight fight between the fraudsters and everyone else. The interests of banks, merchants, regulators, vendors, and consumers clash in interesting ways; the outcome will not just be determined by how the fraudsters adapt to the technology, but by a complex tussle over who pays for the upgrade and who enjoys the benefits. Fraud savings are not the biggest game in town; while fraud costs the U.S. $3–$4 billion, interchange fees are an extraordinary $30 billion and EMV will likely have an impact on both.

Although U.S. banks are issuing EMV cards now, it will be some time before they start to see a reduction in fraud.

Attacks

Although U.S. banks are issuing EMV cards now, it will be some time before they start to see a reduction in fraud. Cards will still have the magnetic strip and banks will continue to accept magnetic strip transactions because it will take many years to upgrade all the ATMs and point-of-sale terminals. EMV terminals still process unencrypted card numbers, expiration dates, and PINs, so if hacked (as occurred in the late-2013 Target data breach), crimi-

IMAGE FROM SHUTTERSTOCK.COM

nals can steal enough data to perform fraudulent online purchases. Also, as many chip cards still contain a full unencrypted copy of the magnetic strip data, the criminals can steal this. If they can get the PIN too, they can make forged cards and use them at an ATM.

EMV also introduces some new vulnerabilities. The first-wave EMV cards in the U.K. were cheap cards capable only of Static Data Authentication (SDA), where the card contains a certificate signed by the bank attesting the card data is genuine. Since this certificate is static, it is trivial to copy it to a counterfeit chip, which can be programmed to accept any PIN—a so-called "Yes-card." Criminals exploited this flaw at a small scale in France, but elsewhere it was not a serious problem. The Yes-card attack can be defeated by online transactions where the merchant contacts the bank to verify the card computed a correct message authentication code on the transaction data. (This uses a key shared between the card and the issuing bank, so the merchant must be online for the code to be checked.) More modern EMV cards also support Dynamic Data Authentication (DDA), which uses asymmetric cryptography and defeats the Yes-card attack even for offline transactions. It is likely that U.S.-issued EMV cards will support DDA and the vast majority of transactions will be online anyway, so the Yes-card attack is unlikely to be a major issue in the U.S.

A much more serious type of fraud in the U.K. was tampering with chip-and-PIN terminals to record card details and customer PINs. Although terminals were certified to be tamper resistant, they were not, and the certification process was seriously flawed.[3] Criminals were able to modify terminals on a large scale to collect customer details as the card sent them to the terminal, and the PIN entered by the customer as it was sent to the card for verification. Because most U.K. cards stored a copy of the magnetic strip on the chip, criminals could then make fake magnetic-strip ATM cards. Before the use of chip and PIN in the U.K., customers signed for store transactions and PINs were only used at ATMs, so tampering with a store terminal did not yield enough information to withdraw cash from an ATM. Chip and PIN changed that: as merchants started accepting PINs at the point of sale, card forgery became easier and more prevalent. As Figure 1 shows, counterfeit card fraud went up after EMV was deployed, and took five years for it to fall back to the previous level. So U.S. banks can expect a lot of attacks using compromised or counterfeit terminals until they can start turning off magnetic-strip fallback mode.

Another attack we worried about in the early days of EMV was the "relay attack."[2] This exploits the fact that while the card authenticates itself to the merchant terminal, the customer does not know which terminal the card is communicating with. If a customer inserts her card into a fake terminal, it can relay a transaction with a quite different terminal. So a crook, Bob, can set up a fake parking meter in New York, and when an unwitting cardholder Alice uses it, Carol (who is colluding with Bob) can stroll into Dave's jewelry store in San Francisco and buy a diamond using a fake card connected to the reader in the parking meter. The poor cardholder thinks she paid $20 for a parking space, and gets a statement showing she spent $2,000 in a store she never visited. The counterfeit card inserted into the genuine terminal simply relays the transaction back to the genuine card via the fake terminal (see Figure 2). While there is no real defense against the relay attack, it does not scale well, so is likely only going to be used against high-value targets.

A more serious vulnerability is the No-PIN attack.[5] In this case, a criminal who has stolen a card but does not know the correct PIN can put a small electronic device between the stolen card and the terminal and use it with any PIN he likes. The device tricks the card into believing it is doing a chip and signature transaction while making the terminal believe the card accepted the PIN that was entered. This attack works against all types of cards, and even for online transactions. Fixing it properly would require a change to the EMV protocol, which would take years to agree. In the interim, it is often possible for the card-issuing bank to detect the attack by carefully comparing the card's version of the transaction with the terminal's. So far, it appears only one U.K. bank is trying to do so. In the meantime, French criminals have been caught exploiting a more sophisticated variant of this attack in the wild.

The latest family of attacks, seen in the last two or three years in Spain, exploits a classic cryptographic vulnerability—the way in which EMV systems generate and use random numbers. When a terminal initiates an EMV transaction, it sends the card not just the date and the amount but a random number, so that each transaction is different and a crook cannot simply replay old transactions. However, there are two flaws in this system. The first is an implementation flaw: it turns out that some ATMs generate predictable "random" numbers, so an attacker who has temporary access to someone else's card (say, a waiter in a criminal-owned restaurant) can calculate an authentication code that he can use at some predictable time in the future at a known ATM. Worse, there is a design flaw in that the terminal does not transmit its choice of random number to the bank in an authenticated way. This means, for example, that if a terminal is running malicious software, it can harvest from a customer's card a series of authentication codes that it can then use to make extra transactions in the future, and it can fix up the random numbers in the protocol so that the issuing bank does not notice anything suspicious.[1] This is a serious attack because it can scale; a crime gang that managed to install malware

Figure 1. U.K. card fraud by category during and since the introduction of EMV. (Source: U.K. Payments Administration.)

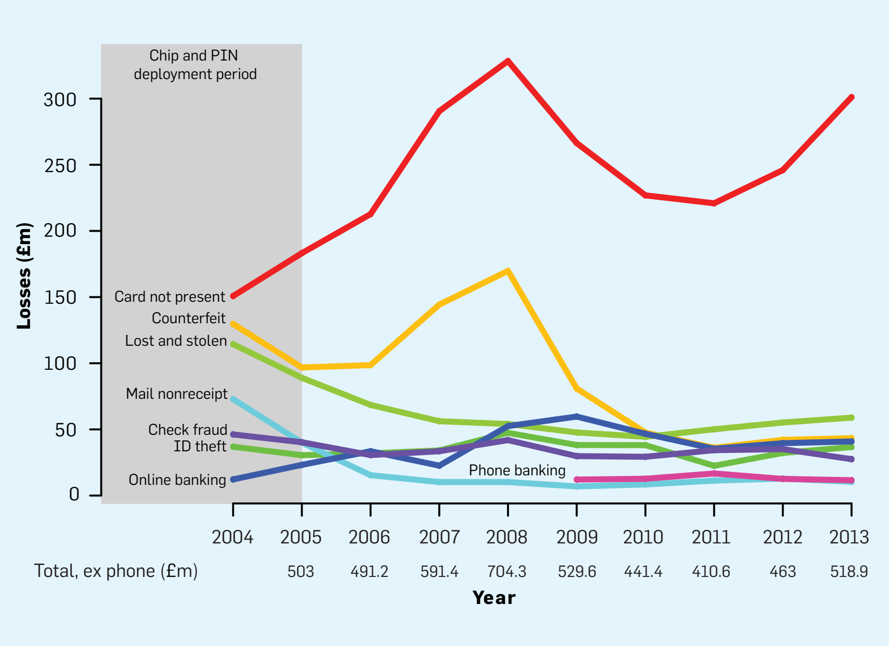
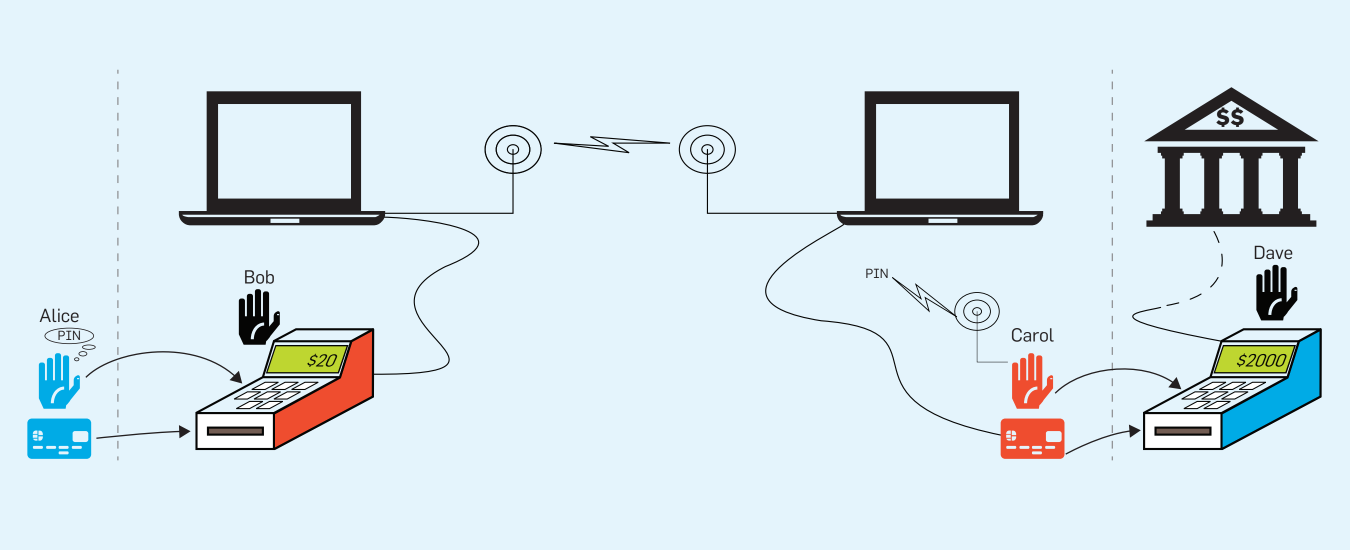


Figure 2. The relay attack.



on a number of legitimate terminals (as happened in the Target case) could harvest authentication codes to authorize large numbers of transactions at businesses under its control.

Finally, the elephant in the room with EMV deployment is card-not-present (CNP) fraud (Internet, phone, and mail-order purchases). Although CNP fraud was low when EMV started to be deployed in the U.K., it had grown to over half of U.K. card fraud by the time the rollout was complete. EMV does almost nothing to stop CNP fraud (cards were never designed to be connected to customer PCs, even if smartcard readers were to become prevalent) and so the crooks' initial reaction to the EMV deployment was just to take their business online, as we can see from the graph in Figure 1. U.S. banks would be well advised to invest in further measures to mitigate CNP fraud rather than putting their entire security budget into deploying EMV. EMVCo (the consortium that maintains the EMV standard) has already started work on a "tokenization" specification, allowing CNP transactions to be performed with limited-use tokens (in effect, one-time credit-card numbers) rather than a static credit-card number, so reducing the damage resulting from merchant data breaches or malware on customer PCs. Tokenization has almost nothing to do with EMV chips, but rather than setting up an new industry body, the banks have drafted in EMVCo to deal with it.

The Business Battleground

When credit cards were first introduced by Diners Club in the 1960s, they had high fees; typically the merchant paid the bank 6% or more of transactions. The emergence in the 1970s of the Visa-MasterCard duopoly stabilized things with standard contracts for banks and merchants, technical standards so their computers could swap data, and standard fees at 2.5% for credit cards and 1.5% for debit. This enabled a huge expansion of the industry, and cards became the standard way to pay for items costing more than a few dollars. Card transaction processing has become a huge revenue generator for the banking industry, especially as the clunky old addressograph machines for taking paper imprints were replaced by cheaper online systems, and as card payments spread online too. Many merchants see the card industry as an exploitative cartel, in need of trustbusting or competitive innovation. In 2005, merchants filed a class-action suit against Visa and MasterCard; a settlement in 2013 lowered fees by 0.1% and allowed merchants to charge customers the higher costs of credit-card transactions (which they already do in Europe). There has also been legislative action, with the Durbin amendment to the Dodd-Frank bill empowering the U.S. Federal Reserve to write the rules for fees on debit card transactions.

> **The most widespread problem that will be encountered by cardholders is likely to be in dispute resolution.**

The sums involved are large. A retailer like Walmart, for example, takes over $200bn in credit-card sales; if these customers could be moved to PIN-based debit card transactions, that would save $2bn in fees. So some retailers have strongly supported the move to EMV. At the same time, the versions of the EMV protocol being introduced in the U.S. to support contactless payments (such as where your credit card becomes an app on your NFC mobile phone) are designed to make it more difficult for merchants to move customers to debit card payments. These market dynamics are unlike those seen in Canada or Europe, where the banking industry motivated merchants to install EMV terminals by means of a "liability shift": the banks changed their terms and conditions so that merchants were charged the cost of all customer disputes where a PIN was not used. Where a PIN was used, the banks would then pass the liability on to the cardholder, saying "Your chip was read and your PIN was used, so you must have been negligent or complicit." Such a liability shift would be more difficult in the U.S. because the retailers' lobby is as powerful as the banks, and because consumer protection is better entrenched in U.S. regulation.

Yet consumer protection may be undermined in a multitude of ways. One example is the protocol used to decide how to authenticate the cardholder. According to the EMV standards, each card has a cardholder verification method (CVM) list that states a preference such as 'first, signature; then PIN'; the terminal should read this and use the highest-ranked method it supports. We would expect to see aggressive retailers programming their terminals to insist on a PIN whenever it is supported, if (as we expect) the fee or liability for a PIN-based transaction is lower. In fact we have encountered cases where merchants have simply lied to the banks about the method used. One fraud victim whose card was stolen while he was on vacation in Turkey was denied a refund for a charge made to his card because the merchant reported it as PIN-based; he managed to get a copy of the receipt for the purchase and discovered the thief had in fact signed for the goods. If you wish to avoid this sort of problem, it is prudent to demand a card that only supports chip-and-signature.

Indeed, as the U.S. will be the first country with a mixture of chip-and-PIN and chip-and-signature cards in issue, we should be able to learn a lot from the crime figures after a few years. And this is not just about fraud, but robbery too. Chip-and-PIN cards are typically capable of offline PIN verification, and European banks have issued millions of card readers that enable cardholders to compute authentication codes for online banking. These readers can be used by muggers to check whether a victim is telling the truth when they demand his PIN as well as his cards; in one unfortunate case, two French students were tortured to death by robbers.

The most widespread problem that will be encountered by cardholders is likely to be in dispute resolution. One of the problems highlighted by the experience in Europe is the lack of suitable tools for courts, arbitrators, and even front-line dispute resolution staff

The bad news is that the interests of banks, merchants, vendors, cardholders, and regulators diverge in significant ways.

in banks. When disputes arose with magnetic-strip cards, the consumer typically got the benefit of the doubt as these were widely known to be forgeable. EMV systems, on the other hand, create large amounts of log data that appear to be impressive but are often not understood, and can sometimes be the result of forgery by merchants (as in the Turkish case) or by malware on merchant terminals (as in the recent Target case, which would likely have been unaffected by the move to EMV). Also, the move from signature to PIN verification shifted dispute resolution in the banks' favor. Any forged signature will likely be shown to be a forgery by later expert examination. In contrast, if the correct PIN was entered the fraud victim is left in the impossible position of having to prove that he did not negligently disclose it.

The main lesson to be learned from these experiences is that the collection, analysis, and presentation of evidence is a function that needs to be specified, tested, and debugged like any other. Simply dumping many pages of printout on a court and leaving it to an expert to pore through the digits, comparing them with EMV manuals, is not a robust way to do things; often the necessary evidence is not even retained. The forensic procedures also need to be open and transparent to stand up in court, and their governance needs to be improved; this problem cannot just be left to a disparate vendor community.[4] Here, some guidance from the Fed would be welcome.

Conclusion

The EMV protocol is not a rigid way of doing card payments so much as a toolkit with which banks can build systems that can be pretty secure, but which can also be pretty awful. There is good news, and bad news. The good news is that EMV systems have been deployed in Europe for 11 years now, and there is a lot of experience to build on. Almost everything that could go wrong, has gone wrong: several protocol flaws that allowed attacks nobody had anticipated; tamper-resistance that did not work; certification schemes that turned out to be a sham; and evidence-collection systems that were not fit for purpose. These should not just be academic case studies for security engineering classes, but should be studied by engineers who want to build robust payment systems.

The bad news is that the interests of banks, merchants, vendors, cardholders, and regulators diverge in significant ways. In Europe, many failures were due to banks dumping liability on merchants and cardholders, who were in no position to defend themselves. In the U.S., the dynamic is different and more complex, with the main fight being over the interchange fees the merchants pay the banks for processing their transactions. These fees are an order of magnitude greater than the fraud is, so we may find that the security of the system will be a side effect of the project rather than its main goal. The details may be fought over for years to come in the courts and by lobbyists in Washington, D.C. C

References

1. Bond, M., Choudary, O., Murdoch, S.J., Skorobogatov, S., and Anderson, R. Chip and skim: Cloning EMV cards with the pre-play attack. In *Proceedings of the IEEE Symposium on Security and Privacy* (San Jose, CA, May 18–21, 2014).
2. Drimer, S. and Murdoch, S.J. Keep your enemies close: Distance bounding against smartcard relay attacks. In *Proceedings of the USENIX Security Symposium* (Boston, MA, Aug. 6–10, 2007).
3. Drimer, S., Murdoch, S.J., and Anderson, R. Thinking inside the box: System-level failures of tamper proofing. In *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, CA, May 18–21, 2008).
4. Murdoch, S.J. and Anderson, R. Security protocols and evidence: Where many payment systems fail. In *Proceedings of Financial Cryptography and Data Security* (Barbados, Mar. 3–7, 2014).
5. Murdoch, S.J., Drimer, S., Anderson, R., and Bond, M. Chip and PIN is broken. In *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, CA, May 16–19, 2010).

Ross Anderson (Ross.Anderson@cl.cam.ac.uk) is Professor of Security Engineering at the Computer Laboratory, University of Cambridge, U.K.

Steven J. Murdoch (Steven.Murdoch@cl.cam.ac.uk) is a Royal Society University Research Fellow at the Computer Laboratory, University of Cambridge, U.K.

DOI:10.1145/2602322

Phillip G. Armour

The Business of Software
Owning and Using

On vital and supporting systems.

Modern companies have systems they need to use and systems they need to own. But there seems to be some loss of focus in the management of these two categories of capability and this may be quite dangerous to the future of the enterprise.

In his seminal work *Post Capitalist Society*[1] the late management guru Peter F. Drucker noted the world has left the era of capital and has now entered the era of knowledge. The cash and hard asset economy is receding and the knowledge economy is on the ascendant. Money is still important; possessing a mound of cash will assist you in consumption, but it may not help much in construction, in building things, in acquiring knowledge. According to Drucker, the critical success factor these days is knowledge: the ability to store and employ knowledge and the ability to create and operationalize it.

Trends and Means

In dealing with a number of situations, companies, and government agencies in the last few years, I have noticed a few trends that are quite disturbing. Two of these trends are contradictory and they relate to how companies build and use computer systems.

If we are indeed moving into the knowledge economy, where knowledge is the defining and limiting asset of the world, we are confronted with the issue of where to keep the stuff. As I have noted in my previous *Communications* columns,[a] we are now moving all the knowledge of the human race from the media of brains and books into an executable software form.

If knowledge is the key asset of the future, then the ability to create or discover that knowledge is the key wealth-generation activity. Since that knowledge will be stored mostly in software, software development should (and will) become the driving force of the new economy. But do companies and countries understand this? If they do, then do they behave as if this capability is the engine of growth that will allow them to flourish in the future? In many cases, it appears not.

If knowledge is the key asset of the future, then the ability to create or discover that knowledge is the key wealth-generation activity.

What You Use, What You Own

There are two kinds of operational capability that companies need. They involve systems that execute "supporting" knowledge and systems that execute "vital" knowledge.

Supporting knowledge is the operationalized knowledge that companies require to support their business. Companies must pay their employees and pay their bills so they need access to payroll and accounts payable systems or services. But few companies need to know how to build these systems. The same is true for many types of capabilities in many types of companies—they need to be able to *use* the knowledge in these systems, they do not need to *own* it.

Vital knowledge is the operationalized knowledge of what the company does for a living, why it exists. This knowledge, how it is obtained, how it is built upon and, most importantly, how it is operationalized (in the software form) is *the most critical asset* for many companies. Organizations need to own this knowledge in all its forms, especially the executable ones. But many companies do not realize this.

A New Life on Lease

Years ago I worked with a transportation equipment leasing company. Their primary function was predicting the equipment leasing needs of their customers, forecasting and managing the

[a] Starting with my first *Communications* "The Business of Software" column: P.G. Armour, The case for a new business model. *Commun. ACM 43*, 8 (Aug. 2000), 19–22; DOI: 10.1145/345124.345131.

availability of resources to match those needs, and synchronizing these needs and resources. In a rapidly changing event-driven marketplace, the optimization of these functions was the key to servicing customers and the key to profitability. Even slightly better optimization could pay off handsomely and would allow the company to outperform their competitors and to thrive. To do this optimization they used a combination of experienced schedulers and sophisticated computer systems driven by linear programming algorithms. Around the time I worked with them, this company was seriously looking at outsourcing its HR systems. Fair enough. But it was also looking at outsourcing the development of the next generation of its equipment leasing optimization systems. The first option was ok—they needed to use an HR system and it made no sense to spend scarce company resources to build one. The second option was not ok—leasing equipment optimization was what this company did. It was their key skill and their market differentiator. Outsourcing this development would essentially be paying someone else to learn their business better than the owners. It is a very dangerous strategy. Ultimately the company chose to enter into a collaboration agreement with a small development firm to build the next-generation optimization system and so retained some ownership of its vital knowledge. But not all businesses do this.

Abdication?

Twenty or more years ago many businesses bravely stepped up to the challenges of the day and built large, complex, and valuable systems that operationalized their vital business knowledge. Now those systems are old and they are running out of gas. They need to be updated to incorporate the knowledge of the modern marketplace and technology infrastructures, to make the systems extensible and scalable, to incorporate new distribution channels and strategic partnerships. Long-established companies are struggling with this challenge and some are retreating from it. Perhaps they have tried revamping these vital assets but were unsuccessful. Maybe they attempted the software process and productivity improvements recommended by internal champions, vendors, and industry gurus but could not seem to make them work or make them stick. Perhaps they reached for a package-driven solution but could not tune it sufficiently to make it a market differentiator. But some companies simply threw up their hands in despair and chose to pay vendors to build their vital systems.

This outsourcing approach amounts to an abdication of the task of extending and operationalizing their vital business knowledge. In the face of the increasing challenges of complexity, technology, market speed, and the perceived difficulty of acquiring skilled staff, these companies are giving up. This seems to be most prevalent in large, hierarchical enterprises where cost and profit are the driving forces coupled with a lack of understanding of the knowledge economy and what these systems really mean.

This is sad and it is dangerous. In their pursuit of a low-cost "solution" or perhaps in pursuit of an easy management life, these companies may be surrendering their key assets and their futures.

Software development is both a learning and a creative activity.

Bespoken For

At the other end of the spectrum, there are companies that, intentionally or accidentally, slip into the practice of building systems they have no business building. While they are boldly stepping up to the challenge of specifying, designing, and building the systems that operationalize what they do for a living, they are also being sidetracked into building things that are not vital knowledge, that are way off their mainstream business functions. Developers may argue they need custom "bespoke" infrastructure,[2] test management systems, development environments, data access methods, even programming languages. They convince the business resourcing decision makers of the inadequacy of available solutions and are given the go-ahead to build their own.

This approach is seductive. The bespoke solution theoretically supports the core business function, it seems to leave the development group in charge of their own development destiny and the hope is that the solution will be optimal for the business. But it can be a misguided and expensive investment and it is often driven by a quite different and more personal agenda.

Learning and Creating

Software development is both a learning and a creative activity. These two forces—to learn and to create—are among the strongest motivators of software engineers, particularly if they have the chance to learn and create something "new." New to them that is. The fact that others have already learned and created the executable result of that learning may not be a big consideration to the developers as they pitch this idea to their executives.

This "grow-your-own" approach seems to be most prevalent in organizations that are technology driven, particularly those that have had innovative technology successes in the past. But when it deflects from operationalizing the primary business knowledge it is a bad thing.

Own Knowledge

The differentiation into vital and supporting knowledge is not as simple as it might seem and I will explore this issue in more depth. But as companies navigate their way into this new era of executable knowledge, they surely need to know what they need to use and know what they need to own and manage accordingly.

If they want to survive in the Age of Knowledge, that is. C

References

1. Drucker, P.F. *Post Capitalist Society.* Harper Business. 1993.
2. Spinellis, D. Bespoke infrastructures. *IEEE Software 31*, 1 (Jan./Feb. 2014), 23.

Phillip G. Armour (armour@corvusintl.com) is a senior consultant at Corvus International Inc., Deer Park, IL, and a consultant at QSM Inc., McLean, VA.

DOI:10.1145/2602323

Dinei Florêncio, Cormac Herley, and Adam Shostack

Privacy and Security
FUD: A Plea for Intolerance

Relying on dubious claims can cause researchers to focus on the wrong questions and organizations to misdirect security spending.

Even a casual observer of computer security must notice the prevalence of FUD: non-falsifiable claims that promote fear, uncertainty, or doubt (FUD). We are bombarded with warnings of digital Pearl Harbors, the unstoppability of online hackers, and accounts of a cyber-crime problem that is said to rival the drug trade.

FUD sometimes masquerades as useful information though it is often "not even wrong," in the sense of making no clear claim that can be checked: exact figures for undefined quantities, dollar estimates based on absurd methodology, and astonishing facts that are traceable to no accountable source. FUD provides a steady stream of factoids (for example, the raw number of malware samples, activity on underground markets, or the number of users who will hand over their password for a bar of chocolate) the effect of which is to persuade us that things are bad and constantly getting worse. While the exaggeration of threats hardly began with computer security, the field has certainly made FUD its own.

It may seem innocent enough to exaggerate in the service of getting people to take security more seriously; but we believe reliance on factoids leads government and industry to spend wastefully and researchers to focus on the wrong questions. The scale of the FUD problem is enormous, and we argue it prevents the establishment of security as a more scientific research discipline.

GLOBAL FEAR
HEARTBLEED'S IMPACT
Margin of Error
SCANDAL
39% of internet users have changed passwords or canceled accounts
6% think their personal information was swiped
CYBERATTACKS HIT THE UNITED STATES
COSTS
Depending on when it ... the first major cyberw... the United States could be a one-in-2 million incident or a one-in-10-trillion incident...
How Many Last Year?
Expectations Look Dim
ON THE RISE
CYBER ATTACKS
FEAR
in 855 data breaches they examined
Target Small Business
Rates
What's Next?
HEAVILY TRAFFICKED
71% occurred in businesses with fewer than 100 e
YOU PREPARED?
Crisis Ahead
Record Number Of Cyber Attacks

What's Wrong with an Illustrative Story?

Offering information that is dubious, false, or vague creates avoidable confusion. Through creating the illusion that we understand when we do not and by injecting false facts, FUD oversimplifies complex questions, hindering our ability to grasp things that might actually be simple.

FUD makes it more difficult to form a coherent picture of the world. While the number of malware samples seems interesting, it says nothing about the success of that malware at infecting systems. Happenings on underground markets are certainly interesting, but activity does not translate into dollars at any fixed rate. Do we know if the passwords that people trade for chocolate are real or made up? These details matter, and their absence hinders understanding. Much FUD comes in the form of factoids, which, of course, can be inconsistent, both with each other and with what else we know of the world. Who exactly lost a trillion dollars? Where are all the cybercrime bil-

lionaires? If cybercrime is so bad and people so careless why doesn't everyone have all of their money stolen every day? Purveyors of factoids make no effort to resolve these or other contradictions. Offered in isolation and selected for effect, FUD claims simply perpetuate uncertainty even on questions where clear answers might be possible.

FUD makes resource allocation difficult. Exaggeration might seem a harmless (or even necessary) tool: a shortcut to the right conclusion when the long way around seems too laborious. However, FUD does not just amplify, it distorts. When we inflate one threat we have to inflate others, as anything exaggerated 10x, seems small relative to things exaggerated 1,000x. Nothing is so small that it cannot be made to look enormous, nothing is so big that it cannot be completely drowned out by the clamor to do something about some other inflated threat. We end up not being able to distinguish urban legends from real threats.

Finally, bad data drives out good. FUD supplies bad information in places where good might be possible. In addition to decreasing the signal-to-noise ratio, it makes it less likely that more scientific measurements will ever be carried out. If everyone already "knows" that baroque password policies and 90-day expiration rules reduce harm, then it is less likely that experiments will test these claims. Once codified as a best practice and followed as a matter of course, the opportunity for observational experiments is limited. Unexamined assumptions thus remain unexamined, and errors that might have been caught persist. The hodge-podge of rules governing passwords have, until very recently, seen little serious research, but receive regular reinforcement in the form of factoids and anecdotes that confirm existing biases. Thus, FUD ensures not merely that the true state of affairs is uncertain, but that it will remain so.

FUD Is Not a Victimless Crime

Those who produce good data suffer in a world that tolerates FUD. Good measurement work is possible[1,2] but expensive; those who produce it suffer if they must compete with FUD for attention. They are robbed of novelty if we feel "we already know" what the landscape looks like based on factoids and FUD. Doing high-quality studies is a losing proposition if those who consume information are indifferent to the care with which it is produced.

It is difficult for a field to advance when unsubstantiated claims circulate unchallenged.

FUD erodes the ability to spend scarce resources sensibly and allocate effort where it will do the most good. Indeed, in exaggerating uncertainty and danger, a main effect of FUD is to thwart the ability of defenders to spend efficiently. Thus everyone who wants good data to make decisions suffers because of FUD.

Finally, research suffers because of FUD. The professional credibility of computer security is impacted as ubiquitous FUD creates the impression of an unscientific field. Is cybercrime a trillion-dollar problem and larger than the global drug trade? Is intellectual property theft "the greatest transfer of wealth in history" or grossly overblown? Is cyber-war the "greatest existential threat we face" or "the new yellowcake?"

It is difficult for a field to advance when unsubstantiated claims circulate unchallenged. The chances of placing security on a more scientific footing seem remote if we cannot agree on what counts as knowledge. Are 99% of exploits really due to known vulnerabilities? Are 70% of compromises actually due to insiders? Explaining what has been observed and making predictions is always difficult, but it seems hopeless if we cannot agree on what is known. FUD obstructs the establishment of a more systematic approach to security problems.

Why Are We Telling You This?

Surely nobody argues for FUD. While security is awash in scare stories and exaggerations, members of the research community (and *Communications* readers) certainly are not directly responsible for this state of affairs. However, as in many fields, "in order for evil to flourish it suffices that good men do nothing." And flourish it certainly does: numbers whose quality is "below abysmal" get repeated by policymakers[4] and trillion-dollar cybercrime numbers become the conventional wisdom. The answer to "How are we doing in security?" is, to quote Viega,[7] "we have no clue." FUD could not achieve this without the acquiescence of many. It is this aspect that we wish to address.

Why is there so much FUD? Reuter[5] suggests that certain conditions favor the spread of fabulist claims and "mythical numbers": the presence of a constituency interested in having the numbers high and the absence of a constituency interested either in having them low or accurate. Drug and crime statistics, for example, can easily become mythical: enforcement agencies have budgets that depend on the numbers being high, but no group is correspondingly interested in understating the numbers. The same one-sided bias applies in our domain. Unlike global warming or the dangers of genetically modified foods, where lobbies exist on both sides of the issue, many in security have the incentive to exaggerate dangers and few if any gain by understating them or ensuring accuracy.

The upward incentives may be beyond our ability to change. However, we can be the constituency that demands accuracy. In our roles as authors, practitioners, and members of the research community we can put out the unwelcome mat for unsubstantiated claims. It is up to those of us in security research to avoid taking shortcuts in making the case for what we do. We can refuse to cite questionable reports, vague claims, or outlandish dollar estimates. As program committee members and reviewers we can ask our colleagues to aspire to a higher standard also. We do not accept sloppy papers, so citing dubious claims (which are simply pointers to sloppy work) should not be acceptable either. An impressive collection of rationalizations is available to excuse the use of unreliable information: data is difficult to get, cybercrime may be underreported (how would one measure that?), and we do face active adversaries. However, unless we resist these temptations we look like a community that responds to uncertainty by lowering standards.

The broader computer science community also has a vital role to play. To have influence, FUD needs to spread. To spread widely it needs the efforts of many people. If they circulate unchallenged, after a while bad numbers come to be accepted as good; unless someone objects, the more they are repeated the more they become part of the consensus view. In this way numbers with little basis in reality[3,6] shape priorities and influence policy.[1,4] Anyone can help halt that process by refusing to forward, quote, or tweet claims that do not seem to add up. This need not be difficult: FUD requires an unskeptical audience and does not fare well under scrutiny. Performing even basic sanity tests and asking "where did that number come from?" or "how would you measure that?" is often all it takes.

Security has many difficulties, but the field has no problem that FUD does not make worse. It will not just go away, but we can help stop the spread. We can make it more expensive to spread FUD than good information by challenging FUD claims every time we hear them.

References

1. Anderson, R. et al. Measuring the cost of cybercrime. In *Proceedings of WEIS 2012.*
2. Bonneau, J. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. *IEEE Security and Privacy* (Nov. 2012).
3. Florêncio, D. and Herley, C. Sex, lies and cyber-crime surveys. In *Proceedings of WEIS 2011.*
4. Maas, P. and Rajagopalan, M. Does cybercrime really cost $1 trillion? *ProPublica* (Aug. 1, 2012); http://www.propublica.org/article/does-cybercrime-really-cost-1-trillion.
5. Reuter, P. The (continued) vitality of mythical numbers. *The Public Interest*, 1987.
6. Shostack, A. and Stewart, A. *New School Security.* Addison-Wesley, 2008.
7. Viega, J. Ten years on, how are we doing? (Spoiler alert: We have no clue). *IEEE Security and Privacy* (Nov. 2012).

Dinei Florêncio (dinei@microsoft.com) is a researcher in the Multimedia, Interaction and Communication group at Microsoft Corporation, Redmond, WA.

Cormac Herley (cormac@microsoft.com) is a principal researcher in the Machine Learning Department at Microsoft Corporation, Redmond, WA.

Adam Shostack (Adam.Shostack@microsoft.com) is a principal program manager on Microsoft's Trustworthy Computing Team in Redmond, WA.

Calendar of Events

June 16–19
The 12th Annual International Conference on Mobile Systems, Applications, and Services,
Bretton Woods, NH,
Sponsored: SIGMOBILE,
Contact: David Kotz,
Email: kotz@cs.dartmouth.edu

June 16–20
ACM SIGMETRICS/ International Conference on Measurement and Modeling of Computer Systems,
Austin, TX,
Sponsored: SIGMETRICS,
Contact: Sanjay Shakkottai,
Email: shakkott@austin.utexas.edu

June 21–25
Innovation and Technology in Computer Science Education Conference 2014,
Uppsala, Sweden,
Sponsored: SIGCSE,
Contact: Asa Cajander,
Email: asa.cajander@it.uu.se

June 21–25
Designing Interactive Systems Conference 2014,
Vancouver, Canada,
Sponsored: SIGCHI,
Contact: Steve Harrison,
Email: sharrison@cal.berkeley.edu

June 22–27
International Conference on Management of Data,
Salt Lake City, UT,
Sponsored: SIGMOD,
Contact: Curtis Dyreson,
Email: curtis.dyreson@usu.edu

June 23–26
ACM Web Science Conference,
Bloomington, IN,
Sponsored: SIGWEB,
Contact: Filippo Menczer,
Email: fil@indiana.edu

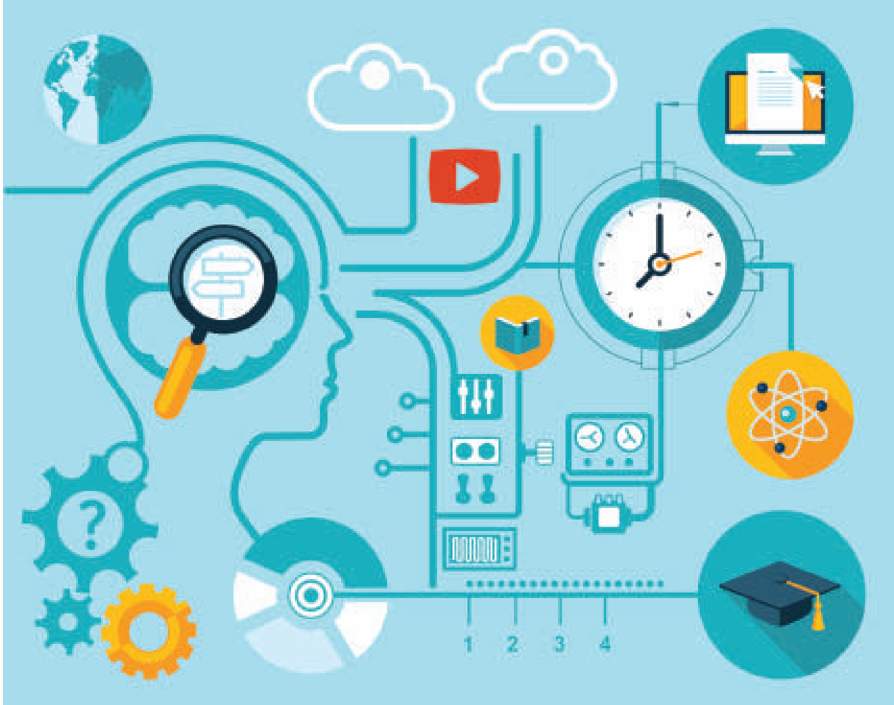
June 23–27
The 23rd International Symposium on High-Performance Parallel and Distributed Computing,
Vancouver, Canada,
Sponsored: SIGARCH,
Contact: Beth A. Plale,
Email: plale@cs.indiana.edu

DOI:10.1145/2602324

Peter J. Denning

The Profession of IT
Avalanches Are Coming

Computing technology has generated conditions for radical transformations of jobs and professions—including education. How shall we cope?



Have you noticed how profoundly information technology is affecting jobs and professions? Previous waves of technology innovation mostly automated manual work: machines displaced blue-collar workers, leaving relatively untouched the white-collar "knowledge workers" celebrated 50 years ago by Peter Drucker. The current wave of information technology innovation is different. It is automating knowledge work and taking jobs away from the middle class. It is producing the greatest financial returns for the designers and builders of the machines, thus contributing to income inequality that troubles so many people.[2,6]

Ubiquitous mobile computers deeply connected into vast networks of information efficiently automate cognitive tasks. For example, Apple's iPhone can talk, answer spoken questions, and recognize fingerprints. Laws in four U.S. states now authorize Google's driverless car. Google Glass overlays real-time displays onto visual scenes, eerily reminiscent of the Star Trek Borg. Graphics processors make virtual worlds look breathtakingly real. IBM's Blue beats world chess grandmasters and IBM's Watson matches natural language patterns faster than the best human "Jeopardy!" players. Supercomputers now pore through huge databases of phone and email metadata to produce detailed reports of any person's movements—and predict their future movements. Facial recognition software does remarkably well at identifying persons in surveillance videos. None of these things seemed possible two decades ago.

During the recent period of global recession, many businesses and government agencies turned to computer-driven automation to increase productivity with fewer workers. As the recession receded, these organizations did not rehire many of the workers they laid off because the machines had replaced them.

In past eras, people looked to education to help them past the disruptions caused by technology innovation. Education could help displaced workers learn new skills and enter new professions. It could help existing professionals stay a step ahead of technology changes. It could channel young people into emerging professions. But now technology advances are disrupting the education system itself. Can the education system help in the current crisis?

I have no answers to these dilemmas. But several new works have looked deeply into these changes and give some insights in ways to cope. These works portray the coming radical transformations as "avalanches," meaning the conditions are ripe for major disruptions whose timing is totally uncertain. The potential for avalanches creates great uncertainty because our science and technology cannot predict or control the changes. I will focus here on education, which has historically been the savior during technological transformations.

An Avalanche Is Coming

Michael Barber, Katelyn Donnelly, and Saad Rizvi offer an unsettling assessment of how many familiar aspects of the higher education system may be disrupted by unbundling in the coming years.[1] They begin by noting six big trends: globalization of markets for faculty and students, debt crises causing governments to reduce education subsidies, tuition hyperinflation, falling value of formal degrees (median salaries of bachelor's degree holders are down 15% since 2000), ubiquity of free content, and hot competition (for example, MOOCs). These trends have created a climate for third parties to perform individual, traditional functions of a university at a significantly lower cost.

The result is that the 10 traditional functions of university (see the accompanying table) are being unbundled. A student can assemble a package of unbundled functions to approximate a big university at a fraction of the cost.

The MOOC is a highly visible example of potential disruption. This technology suddenly grabbed attention in 2011 when Stanford University made its artificial intelligence course available tuition-free worldwide, and got 160,000 registrants. Entrepreneurs quickly founded new companies and consortia including Coursera, Udacity, and EdX to offer courses from elite universities tuition-free worldwide. These groups have developed platforms to deliver content, interact with students, assess progress, and provide certificates of completion. The promoters see MOOCs as mass-production machines that automate all the processes of a traditional class and scale them up for tens or hundreds of thousands of students. MOOCs have inspired experiments with new modes of learning such as the "flipped classroom" and the "10-minute video." At the same time, the early figures raise questions about some of the promoters' claims. For example, MOOCs have experienced attention-grabbing attrition rates in excess of 90%; is the reach of MOOCs likely to be much less than promoters claim? Many course graduates have already covered the material in another course; are current MOOC offerings too high-level for uneducated people in developing countries? Is it ethical to sell the names of active students to prospective employers? Will the need for faculty and teaching assistants plummet? Are MOOC production studios a good investment?

As interesting as these questions are, MOOCs are a threat in just one of the 10 dimensions. Disruptions are brewing the other nine. Avalanches from other directions are more likely.

Barber et al. argue that universities will have to develop distinctive offers to survive. Few universities fully configured to offer all traditional services will be sustainable. Specialized universities can emerge including elite, mass, niche, local, and lifelong learning. Although the primary goal of education—to transmit a culture to the students—is unchanged, many practices will change.

And Barber et al. argue that the current ranking systems (for example, by *U.S. News and World Report*) are stacked in favor of the elite universities. Only those with large research enterprises and small student-faculty ratios can earn the highest rankings. There is no way to measure excellence in other distinctive categories such as mass education, niches, local service, or continuing education. Barber et al. would like to see that ranking system dismantled.

Surfing Toward the Future

The authors of the surfing report,[4] which I highlighted in my December 2013 column, devoted a whole section to education. Their concern is that education should help prepare people for a fast-changing world where technology enables radical transformations at an accelerating pace. They argue that the education system must be reformed to live up to this challenge.

The biggest change for education is to prepare people to function well in a world replete with uncertainties. The tradition of scientific and technological knowledge gives the illusion that things follow laws, have predictable paths, and can be controlled. But since technology is embedded in human social systems, and its use depends on unpredictable human actions and declarations, the path of technology itself is unpredictable. Yes, we can see gross trends like Moore's Law in the technology; but we cannot say with any certainty which of many future possibilities will turn out. And often some of those possibilities are radically disruptive and unexpected. The same question challenges many countries: How do we want our education system to prepare our young people for a world full of uncertainties?

The authors argue that two great meta-skills, design and entrepreneurship, give the essence of a new response to this question. Design is concerned with understanding the concerns and issues of a community and making proposals to combine existing technologies and components into a means of dealing with those concerns. Designers perpetually interpret the world; they are exploratory, sensitive to symbolism and identity, respectful of reality,

Functions of a traditional university (adapted from Barber et al.[1]).

1. Research	Journal publications, reports, citations, patents.
2. Degrees	Diplomas, certificates, transcripts; brand values.
3. Local connectivity	Economic and social development of region.
4. Faculty	Professors, lecturers, teaching assistants.
5. Students	Full and part time, usually 18–22 years of age.
6. Governance	University leadership, board; processes such as admissions, record keeping, fundraising, alumni services, maintenance.
7. Curriculum	Subject-based courses grouped into degree programs; faculty prepare course content and syllabus; textbooks; reading materials.
8. Pedagogy	Lectures, presentations, seminars, tutorials, project and thesis advising.
9. Assessment	Exams, finals, thesis defenses, project presentations.
10. Experience	Student organizations, co-curricular activities (such as debating or research competitions); extracurricular activities (such as drama, sports), work experience (such as volunteering, internships).

and mindful of ethical consequences of their proposals. Entrepreneurship is concerned with creating offers and transforming markets to take care of concerns. Entrepreneurs also listen for concerns, they create businesses, they build trust, and they are constantly reading the world. Designers and entrepreneurs often work together.

The authors discuss the moods (dispositions) that facilitate the work of designers and entrepreneurs. Topping their list are gratitude for what is already here, care for future generations, and radical hope. Radical hope is a name they borrowed from Jonathan Lear[5] to cope with disruptive transformations: it is a disposition to accept the change, let go of the identity of the past, and invent new offers in the new world. The authors propose that education institutions learn how to cultivate these dispositions.

The authors also discuss at some length the disillusionment of youth at the current education system. The current system focuses on transmitting knowledge in the form of many facts and processes students must memorize. But, not seeing much of it as useful in the world, students react with boredom and indifference. They do not learn the practices and dispositions that will help them cope with uncertainty.

An example of the disconnect can be seen in the way job interviews now work. A decade ago, the degree certificate and a good résumé were the tickets to employment after graduation. But employers have not found much correlation between degrees and success at work. Instead, they have invented their own performance-based interview processes. You demonstrate your knowledge of programming by solving problems during the interview. Young people are smart: they learn how the companies interview and do their own preparation, often outside of school. They wind up thinking that a lot of school is irrelevant.

How and What to Learn

In recent years, the term "knowledge" has narrowed to mean information—the facts and recorded narratives of a field. The term "body of knowledge," common in education circles, denotes an organized compendium of the topics to be covered by a curriculum.

Accelerating technology is creating greater uncertainty, and threatens disruptions of familiar institutions.

But the wisdom of the ages tells us knowledge is much broader than this. Knowledge also includes the ability to perform skillfully, for example, when we say a taxi driver knows the city or a software engineer knows an operating system. It also includes the ability to know a good direction of movement to achieve a goal. We have terms for these three shades of meaning of knowledge: *descriptive knowledge*, or knowing about; *practices*, or knowing how; and *dispositions*, or knowing which direction to move.

Douglas Thomas and John Seely Brown use the terms *homo sapiens* (one who knows), *homo faber* (one who makes), and *homo ludens* (one who plays or experiments) to refer to personifications of these three kinds of knowledge.[3] The latter two are easy to miss since they depend on "tacit knowledge," which is what we know but cannot put in language how we do it. Thomas and Brown say our education system is based on a philosophy that elevates explicit knowledge to the top priority and gives students little opportunity to become skillful at essential practices or develop useful dispositions. They call for a rebalancing of the system, giving equal emphasis to all three.

Their attention to dispositions is novel. A disposition is a tendency to interpret the world and act in it in a certain way. This kind of knowledge must be cultivated. It cannot be presented as facts or learned through practice. Thomas and Brown say the kinds of dispositions needed in the new world are curiosity, questing, connecting, and reflecting. Their list complements Flores's list of design, entrepreneurship, gratitude, future caring, and radical hope. Certainly this list is incomplete. A military school might aim to cultivate a warrior disposition; a business school might aim for a marketing disposition. All the authors praise virtual realities, virtual learning communities, and games as spaces that can encourage these dispositions.

Educators do not seem to have spent much time thinking about how to cultivate dispositions, which are often seen as "values" left to extracurricular activities such as clubs.

What It Means for You

Big changes are afoot. Accelerating technology is creating greater uncertainty, and threatens disruptions of familiar institutions. No one knows when or if the disruptions might occur; they can be as sudden and as devastating as an avalanche.

An avalanche can come to your part of the computing profession if your work is of the kind that faster computers and big data processors can automate.

Your best defenses are in education. Pay attention to the knowledge, practices, and dispositions you have. Find mentors to help you learn new practices. Join learning communities to help you foster new dispositions. Spend a lot more time reading to find out what is going on in the world. Take advantage of new education opportunities such as MOOCs.

References

1. Barber, M., Donnelly, K., and Rizvi, S. *An Avalanche is Coming: Higher Education and the Revolution Ahead*; http://www.pearson.com/avalanche.
2. Brynjolfsson, E. and McAfee, A. *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies.* W.W. Norton, 2014.
3. Douglas, T. and Seely Brown, J. *A New Culture of Learning: Cultivating the Imagination for a World of Constant Change.* Create Space Independent Publishing Platform. 2011.
4. Flores, F. Report of Consejo Nacional de Innovación para la Competitividad. *Surfing Towards the Future: Chile on the 2025 Horizon.* 2013; http://chile-california.org/surfing-towards-the-future-into-counselor-fernando-floress-vision-for-innovation.
5. Lear, J. *Radical Hope: Ethics in the Face of Cultural Devastation.* Harvard University Press. 2008.
6. *The Economist.* The future of jobs; Technology and jobs. (Jan. 18, 2014).

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, CA, is Editor of ACM *Ubiquity*, and is a past president of ACM. The author's views expressed here are not necessarily those of his employer or the U.S. federal government.

DOI:10.1145/2602325

George V. Neville-Neil

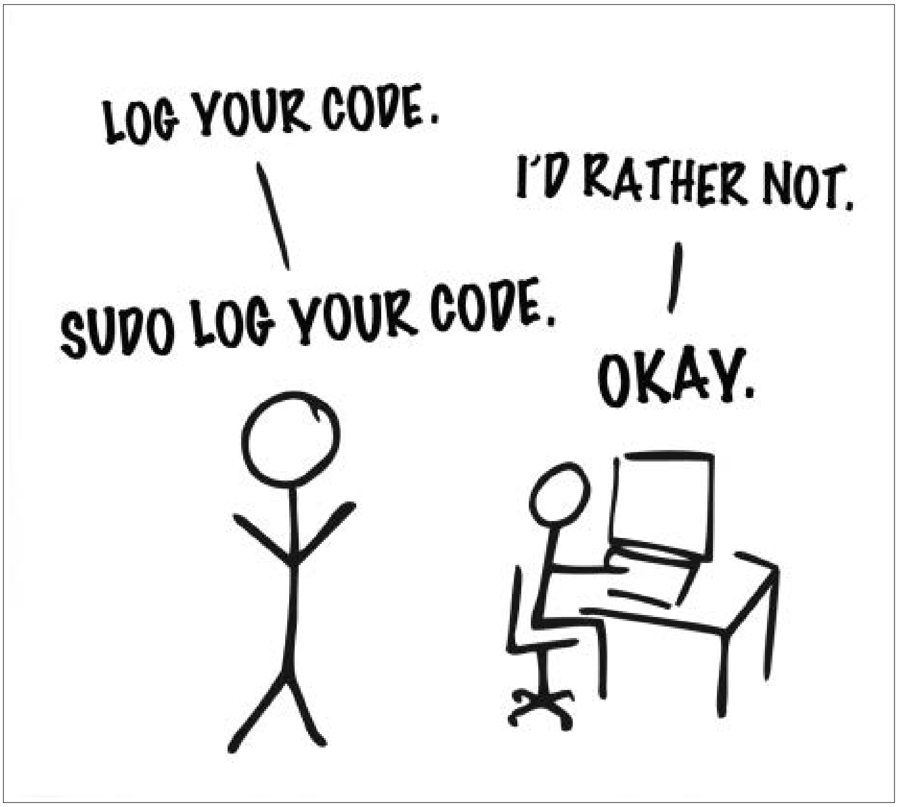
Article development led by acmqueue
queue.acm.org

Kode Vicious
The Logic of Logging

And the illogic of PDF.

Dear KV,
I work in a pretty open environment, and by open I mean that many people have the ability to become the root user on our servers so they can fix things as they break. When the company started, there were only a few of us to do all the work, and people with different responsibilities had to jump in to help if a server died or a process got away from us. That was several years ago, but there are still many people who have rootly powers, some because of legacy and some because they are deemed too important to restrict. The problem is that one of these legacy users insists on doing almost everything as root and, in fact, uses the `sudo` command only to execute `sudo su -`. Every time I need to debug a system this person has worked on, I wind up on a two- to four-hour log-spelunking tour because he also does not take notes on what he has done, and when he is finished he simply reports, "It's fixed." I think you will agree this is maddening behavior.

Routed by Root



CARTOON BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

Dear Routed,
I would like to tell you that you can do one thing and then say, "It's fixed," but I cannot tell you that. I could also tell you to take off the tip of his pinky finger the next time he does this, but I bet HR frowns on Japanese gangster rituals at work.

What you have is more of a cultural problem than a technical problem, because as you suggest in your letter, there is a technical solution to the problem of allowing users to have auditable, root access to systems. Very few people or organizations wish to run their systems with military-style security levels; and, for the most part, they would be right, as those types of systems involve a lot of overkill—and, as we have seen of late, still fail to work.

In most environments it is sufficient to allow the great majority of staff to have access only to their own files and data, and then give a minority of staff—those whose roles truly require it—the ability to access broader powers. Those in this trusted minority must not be given blanket authority over systems, but, again, should be given limited access, which, when using a system such as sudo, is pretty simple. The rights to run any one program can be whitelisted by user or group, and having a short whitelist is the best way to protect systems.

Now we come to your point about logging, which is really about auditability. Many people dislike logging because they think it is like being watched, and, it turns out, most people do not like to be watched. Anyone in a position of trust ought to understand that trust needs to be verified to be maintained and that logging is one way of maintaining trust among a group of people. Logging is also a way of answering the age-old question, "Who did what to whom and when?" In fact, this is all alluded to in the message that sudo spits out when you first attempt to use it:

```
We trust you have received
the usual lecture from the
local System Administrator.
It usually boils down to
these three things:
#1) Respect the privacy of
others.
#2) Think before you type.
#3) With great power comes
great responsibility.
```

Item #3 is a quote from the comic book *Spider-Man*, but Stan Lee's words are germane in this context. Root users in a Unix system can do just about anything they want, either maliciously or because they are not thinking sufficiently when they type. Frequently, the only way of bringing a system back into working order is to figure out what was done to it by a user with rootly powers, and the only way of having a chance of doing that is if the actions were logged somewhere.

If I were this person's manager, I would either remove his sudo rights completely until he learned how to play well with others, or I would fire him. No one is so useful to a company that they should be allowed to play god without oversight.

KV

Dear KV,

I was recently implementing new code based on an existing specification. The documentation is 30 pages of tables with names and values. Unfortunately, this documentation was available only as a PDF, which meant there was no easy, programmatic way of extracting the tables into code. What would have taken five minutes with a few scripts turned into a half day of copy and paste, with all the errors that implies. I know that many specifications—Internet RFCs (Requests for Comments), for example—are still published in plain ASCII text, so I do not understand why anyone would publish a spec destined to be code in something as programmatically hard to work with as PDF.

Copied, Pasted,
Spindled, Mutilated

Other than in mythology, standards and code are not written in stone.

Dear Mutilated,

Clearly you do not appreciate the beauty and majesty implicit in modern desktop publishing. The use of fonts—many, many fonts—and **bold** and underline, increase the clarity of the written words as surely as if they had been etched in stone by a hand of flame.

You are either facing a problem of overwhelming self-importance, as when certain people—often in the marketing departments of large companies—start sending email with bold, underline, and colors, because they think that will make us pay attention, or you are dealing with someone who writes standards but never implements them. In either case this brings up a point about not only documentation, but also code.

Other than in mythology, standards and code are not written in stone. Any document that must change should be trackable in a change-tracking system such as a source-code control system, and this is as true for documents as it is for code that will be compiled. The advantage of keeping any document in a trackable and diffable format is that it is also possible to extract information from it using standard text-manipulation programs such as sed, cut, and grep, as well as with scripting languages such as Perl and Python. While it is true that other computer languages can also handle text manipulation, I find many people doing what you suggest are doing it with Perl and Python.

I, too, would like to live in a world where—when someone updated a software specification—I could run a set of programs over the document, extract the values, and compare them with those that are extant in my code. It would both reduce errors and increase the speed with which updates could be made. The differences might still need to be checked visually, and I would check mine visually out of basic paranoia and mistrust, but this would be far easier than either printing a PDF file and going over it with a pen or using a PDF reader to do the same job electronically. While there are programs that will tear down PDFs for you, none is very good; the biggest problem is that if the authors had thought for 30 seconds before opening Word to write their spec, none would be necessary.

There are many things one can complain about with respect to the IETF (Internet Engineering Task Force), but the commitment to continue to publish its protocols in text format is not one of them. Alas, not everyone had Jon Postel to guide them through their first couple of thousand documents, but they can still learn from the example.

KV

Related articles on queue.acm.org

Debugging on Live Systems
George Neville-Neil
http://queue.acm.org/detail.cfm?id=2031677

Multitier Programming in Hop
Manuel Serrano, Gérard Berry
http://queue.acm.org/detail.cfm?id=2330089

A Plea to Software Vendors from Sysadmins—10 Do's and Don'ts
Thomas A. Limoncelli
http://queue.acm.org/detail.cfm?id=1921361

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and co-chair of the *ACM Queue* editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.

DOI:10.1145/2602326

Charles K. Davis

Viewpoint
Beyond Data and Analysis

Why business analytics and big data really matter for modern business organizations.

Wherever business executives turn these days, someone expounds the merits of business analytics, or some derivative of business analytics like supply chain analytics or marketing analytics or human resources analytics, or even predictive or Web or visual or data or streaming analytics, or any number of others.[5,8] The academic community is also promoting this emerging view of analytics.[6] There has recently been a call for a new professional role, that of the data scientist, to implement and diffuse analytics methodologies into and across organizations.[3] There is even concern there will not be enough of these new professionals to meet the growing demand for this analytics specialty even in the immediate future. So, what does all of this really mean?

Today, businesses are awash in data.[10] In wave after disruptive wave of technological and organizational change, business leaders face a host of powerful forces. For example, information processing has become increasingly more powerful and flexible, with faster and higher-capacity storage and networks. Simultaneously, globalization and other competitive factors have exerted strong pressures to improve efficiencies and effectiveness, and to strengthen business and customer relationships. Each successive stage of this competition requires more data and more analysis to support strategic, managerial, and operational decision-making. This competition, therefore, is driving a quest for more and better analytics technology; and this technology, in turn, helps to make competition even more intense. This cycle results in a confluence of competitive imperatives and technological advancements that interact dramatically. More effective analytics enables a higher level of competition; and higher competition creates further imperatives to make the analytics more effective. Advancing technology creates more competition, which creates more technology, which creates more competition, and so it goes.

Real Change or More of the Same?

Still, as a practical matter, it is not difficult to be skeptical about business analytics.[1,7] This is primarily because what is being touted as 'new' is a set of well-established and already widely used analytical approaches and methodologies which, aside from a few refinements, are not new at all—and have historically on occasion been unreliable or impractical. These new analytics employ essentially the same mul-

```
} else {
if ($id=intval($_GET['id'])){
$query="SELECT * FRO
$res=mysql_query($que
$row=mysql_fetch_asso
foreach ($row as $k => $v
} else {
$row['name']='';
$row['id']=0;
}
include 'form.php';
}
?>

form.php
<form method="POST">
<input type="text" name="nam
<input type="hidden" name="i
<input type="submit"><br>
<a href="?">Return to the list</
</form>

list.php
<a href="?id=0">Add item</a>
<? foreach ($LIST as $row): ?>
<li><a href="?id=<?=$
<? endforeach ?
```

PHOTOGRAPH BY VITALIY NOVITSKY

What dependence upon analytics implies about the ways businesses must now compete is what is truly profound.

tivariate inferential and descriptive statistical methods and mathematical modeling techniques that have long been used by businesses for analyzing data to support instances of complex decision making. For example, correlations, cluster analysis, filtering, decision trees, Bayesian analysis, neural network analysis, regression analysis, textual analysis, and so forth are all in the analytics arsenal, and none of this is particularly new.[2] Furthermore, even with today's most modern techniques and tools, this kind of analytics still has practical limitations. For example, software and data complexities can impede effective analysis, and interpreting the results of complex analyses accurately can be potentially perilously misleading. It is, therefore, difficult to appreciate this latest emphasis on business analytics is anything unusual or different in comparison to the analytical processes that have been routinely employed by a host of serious-minded business decision makers in the past. One might well argue this change appears to be essentially incremental and does not embody any fundamental paradigm shifts.

Another key aspect of business analytics is often called big data.[9] This is characterized by vast collections of variously structured and even unstructured data that, when appropriately rationalized, can provide understanding and insight into various issues that reside embedded within that data.[4] But this is also something that executives have been addressing for a long time. Remember the term "information overload" that was so popular a decade ago? The essential tools and techniques for dealing with big data include database management, especially data warehousing, data mining, dashboards, and associated technologies. These are hardly new constructs in the realm of managing data. Neither are the concepts, for example, of data ambiguity, data filtering, data context, data interpretation, data conversion, or data redundancy. Again, aside from a few progressive data management refinements, it can hardly be argued that any of this is particularly new either.

In theory, big data is different because of the three V's—volume, velocity, and variety.[10] That is, big data consists of expansive collections of data (large volumes) that are updated quickly and frequently (high velocity) and that exhibit a huge range of different formats and content (wide variety). These factors force organizations to pursue increasingly innovative and cost-effective approaches to organizing, processing, and delivering timely information. It is argued that big data represents a real departure from the past. Fair enough; still, this argument is essentially evolutionary. Companies have been experiencing incremental expansions of volumes, velocities, and varieties of data for decades, more recently accelerated by the growth of the Web. Except perhaps for its sheer volume, this does not appear to be anything particularly unusual or unexpected. Nonetheless, cumulative change is only one aspect of this question. The real thrust here involves how this accumulation interacts with and impacts organizational strategies, operations, and controls beyond the technology.

Basic Drivers for Change

So, why is business analytics suddenly drawing so much attention? One point is becoming clear. This is not likely to be just another buzzword that is hyped for a while and then recedes from the limelight quietly. Companies are really concerned about this issue, and there is traction for real change here. Why?

Businesses are experiencing ever-expanding cycles of change caused by the interaction of competitive forces and the harnessing of analytics and big data technologies as essential competitive weaponry. Technologies for collecting, manipulating, transmitting, and analyzing data have been improving for a long time. What is new and different is that these technologies have reached, and are surpassing, a capacity threshold for processing and storing data that is swamping conventional levels of organizational ability to cope with the volumes of data being generated. Business analytics technologies enable organizations to better cope with these new processing realities. This is important because the cumulative reach and scope of the underlying technologies and methods in use today reflect a level of impact on organizations that makes large-scale analytics critical for both sustaining business competitiveness and enhancing day-to-day decision making. It is the reach and scope of these technologies and methods that actually matter here, not whether the analytics and data tools are newly invented or not. This revolution is real and it is permanent.

Why is this happening now? Beyond technological innovations that make it possible to accumulate and process massive amounts of data ever more cost-effectively, the other key concept here is a competitive mandate that businesses continuously improve their decision-making capabilities in order to survive.[6] The consistent, systematic analysis of complex data for decision making enables a company to operate more intelligently at all levels. In particular, the emphasis upon strategic business analytics in recent years has elevated executive expectations and helped to transform the business analytics ideal into a significant competitive force. The application of business analytics methods leads to improvement in an organization's overall decision-making capacity, which enhances its ability to conduct its business intelligently. So, the desire (and accelerating need) to achieve a higher level of organizational intelligence is a prime driver for implementing business analytics.

An Executive Perspective

These arguments resonate well, but there may be a broader explanation for this phenomenon. Since the advent of computing (and networking) as a profession, computing professionals have shared a common vision—that this technology over time is destined to eventually become wholly integrated into every operating and managerial function in every part of every organization. Today's business analytics is a manifestation of that vision. In a very real sense, the usage threshold that has been reached for computing technologies in modern organizations is that of utter dependence upon timely information for basic competitive viability. This business analytics ideal implies a mandate for data collection and manipulation on a grand scale—just to be able to compete in the modern global marketplace. This dependency has evolved from (and is driven by) the relentless march of progress in computing technologies, undoubtedly; but what dependence upon analytics implies about the ways businesses must now compete is what is truly profound. Business analytics and big data are not just marketing hype—or more of the same old statistical analysis and data manipulation methods that have always been around. This is the future. The concept of analytics, as it is understood today, really is new. It is a term that embodies the realization of a historic vision of how computing will challenge and change the world of business, forever and irrevocably, a vision that is now coming to pass under the guise of business analytics.

References

1. Barton, D. and Court, D. Making advanced analytics work for you. *Harvard Business Review 90*, 10 (Oct. 2012), 78–83.
2. Davenport, T.H. and Harris, J.G. The prediction lover's handbook. *MIT Sloan Management Review 50*, 2 (Feb. 2009), 32–35.
3. Davenport, T.H. and Patil, D.J. Data scientist: The sexiest job of the 21st century. *Harvard Business Review 90*, 10 (Oct. 2012), 70–76.
4. Davenport, T.H., Barth, P., and Bean, R. How "big data" is different. *MIT Sloan Management Review 54*, 1, 43–46.
5. Hopkins, M.S., Lavalle, S., and Balboni, F. 10 insights: A first look at the new intelligent enterprise survey. *MIT Sloan Management Review 52*, 1 (Jan. 2010), 22–26.
6. Hsinchun Chen, H. Chiang, R.H.L., and Storey, V.C. Business intelligence and analytics: From big data to big impact. *MIS Quarterly 36*, 4 (Apr. 2012), 1165–1188.
7. Jacobs, A. The pathologies of big data. *Commun. ACM 52*, 8 (Aug. 2009), 36–44.
8. Kiron, D. and Shockley, R. Creating business value with analytics. *MIT Sloan Management Review 53*, 1 (Jan. 2011), 57–63.
9. LaValle, S., Lesser, E., Shockley, R., Hopkins, M.S., and Kruschwitz, N. Big data, analytics and the path from insights to value. *MIT Sloan Management Review 52*, 2 (Feb. 2011), 21–22.
10. McAfee, A. and Brynjolfsson, E. Big data: The management revolution. *Harvard Business Review 90*, 10 (Oct. 2012), 78–83.

Charles K. Davis (ckdavis@post.harvard.edu) is the Cameron Endowed Chair and Professor of Information Management in the Cameron School of Business at the University of St. Thomas in Houston, TX. He was recently a Visiting Scholar at the Center for Transportation and Logistics at the Massachusetts Institute of Technology, which supported this research.

DOI:10.1145/2605205

Article development led by acmqueue
queue.acm.org

Looking at embedded DSLs.

BY ANDY GILL

Domain-Specific Languages and Code Synthesis Using Haskell

THERE ARE MANY ways to give instructions to a computer: an electrical engineer might write a MATLAB program; a database administrator might write an SQL script; a hardware engineer might write in Verilog; and an accountant might write a spreadsheet with embedded formulas. Aside from the difference in language used in each of these examples, there is an important difference in *form* and *idiom*. Each uses a language customized to the job at hand, and each builds computational requests in a form both familiar and productive for programmers (although accountants may not think of themselves as programmers). In short, each of these examples uses a Domain-Specific Language (DSL).

A DSL is a special-purpose language, designed to encapsulate possible computations in a specific domain. In the earlier examples of MATLAB, SQL, Verilog, and spreadsheets, the domains would be scientific modeling, database queries and updates, hardware circuits, and financial computations, respectively. Considering SQL specifically, there is nothing it does that could not be done in Java or C, or any other general-purpose programming language. SQL simply bundles the actions needed to interact with a database into a

ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

usable and productive package, and the language becomes the interface to communicate requests to the database engine.

There are two fundamental types of DSLs. The first is a first-class language, shown in Figure 1(1), with its own compiler or interpreter, and it is often used in its own ecosystem. All the examples mentioned so far fall into this category. The primary difference between the SQL DSL and, for example, Java is one of scope and focus, although sometimes DSLs grow to be as large as general-purpose languages.

The other class of DSL is a language embedded in a host language, as shown in Figure 1(2). Such languages can have the look and feel of being their own language, but they leverage the host language's existing ecosystem and initial semantics. This article is concerned with this second class of DSLs.

Haskell Primer

An embedded DSL (EDSL) is a language inside a language. Haskell,[17] the premier pure functional programming language, is a great host for EDSLs because of flexible overloading, a powerful type system, and lazy semantics. This section provides a terse introduction to Haskell, sufficient to make this article self-contained. It is an extended version of the Haskell primer I gave in 2011 at the International Conference on Engineering of Reconfigurable Systems and Algorithms.[10]

Haskell is all about *types*. Types in Haskell, like those in other languages, are constraining summaries of structural values. For example, in Haskell `Bool` is the type of the values `True` and `False`; `Int` is the type of machine-sized words; `Double` is the type of double-precision IEEE floating-point values; and this list goes on in the same manner as C, C++, Java, and other traditional languages. All of these type names in Haskell start with an uppercase letter.

On top of these basic types, Haskell has two syntactic forms for expressing compound types. First, pairs, triples, and larger structures can be written using tuple-syntax, comma-separated types inside parentheses. Thus, (Int,Bool) is a structure with both an Int and a Bool component. Second, lists have a syntactic shortcut, using square brackets. Thus, [Int] is a list of Int.

Haskell also has other container types. A container that *may* contain one Int has the type Maybe Int, which is read *Maybe of Int*. These container names also start with uppercase letters.

Types can be nested to any depth. For example, you can have a [(Maybe (Int,Bool))], read as *list of Maybe of (Int and Bool)*.

Polymorphic values are expressed using lowercase letters and play a similar role to void* pointers in C and polymorphic arguments in the Java generics facility. These polymorphic values can have constraints expressed over them, using the Haskell equivalent of an object hierarchy.

Finally, a function is written using an arrow from argument type to result type. Thus, in Haskell, a function that takes a list and returns a list is written as: [a] -> [a].

Here is an example of a Haskell function:

```
sort :: (Ord a) => [a] -> [a]
sort []   = []
sort (x:xs) = sort before
   ++ [x] ++ sort after
  where
   before = filter (<= x) xs
   after  = filter (> x) xs
```

This function sorts a list using a variant of quicksort in which the pivot is the first element of the list:

- The first line is the type for sort. This is ∀a, such that a can be ordered (admits comparisons like <=); the function takes and return a list of such a's.
- The second line says that an empty list is already sorted.
- The remaining lines state that a non-empty list can be sorted by taking the first and rest of the list (called x and xs, respectively), sorting the values before this pivot and after this pivot, and concatenating these intermediate values together.
- Finally, intermediate values can be named using the where syntax; in this case the values of before and after.

Haskell is a concise and direct language. Structures in Haskell are denoted using types, constructed and deconstructed, but never updated. For example, the Maybe type can be defined using two constructors, Nothing and Just:

Figure 1. Types of domain-specific languages.

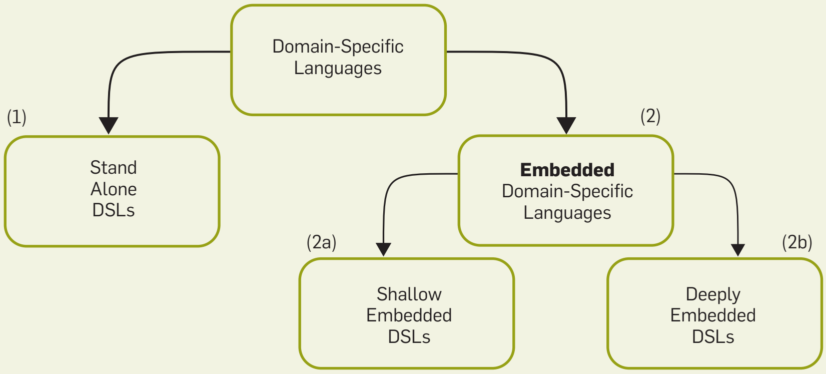


Figure 2. A counter in Kansas Lava.

```
counter :: (Rep a, Num a) => Signal Bool -> Signal Bool -> Signal a
counter restart inc = loop
   where reg = register 0 loop
         reg' = mux2 restart (0,reg)
         loop = mux2 inc (reg' + 1, reg')
```

```
data Maybe where
  Nothing  ::      Maybe a
  Just     :: a -> Maybe a
```

Nothing is a Maybe of anything; Just, with an argument, is a Maybe with the type of the argument. These constructors can be used to construct and deconstruct structures, but there is never any updating; all structures in Haskell are immutable.

It is possible to give specific types extra powers, such as equality and comparison, using the class-based overloading system. The Maybe type, for example, can be given the ability to test for equality, using an instance:

```
instance Eq a => Eq (Maybe a)
 where
  Just a  == Just b  = a == b
  Nothing == Nothing = True
  _     == _     = False
```

This states that for any type that can be tested for equality, you can also check Maybe of the same type. You take the Maybe apart, using pattern matching on Just, to check the internal value.

In Haskell, side effects such as writing to the screen or reading the keyboard are described using a do-notation:

```
main :: IO ()
main = do
  putStrLn "Hello"
  xs <- getLine
  print xs
```

In this example a *value* called main uses the do-notation to describe an interaction with a user. Actually, the do-notation captures this as a structure called a *monad*; purity is not compromised. More detailed information is available on how the do-notation and monads can provide an effectful interface inside a pure language such as Haskell.[18] For the purposes of this article, do-notation is a way of providing syntax and structure that looks like interaction. There are many tutorials on Haskell; the Haskell website, (http://haskell.org) is a good starting point for further reading.

Embedded DSLs

An EDSL is a library in a host language that has the look, feel, and semantics of its own language, customized to a specific problem domain. By reusing the facilities and tools of the host language, an EDSL considerably lowers the cost of both developing and maintaining a DSL. Benefiting from Haskell's concise syntax, the Haskell community—and the functional programming community in general—has taken the ideas of EDSLs and developed a large number of DSLs that provide higher-level interfaces and abstractions for well-understood systems. What follows are two examples of EDSLs: one for automatically generating test cases for software testing; and a second for specifying hardware-circuit behaviors.

Example EDSL: QuickCheck Properties. Consider the challenge of writing test cases—or more specifically, writing the *properties* that test cases need to satisfy:

```
--The reverse of a reverse'd
--list is itself
prop_reverse_twice
   (xs :: [Int])
 = reverse (reverse xs) == xs
```

In this example, prop_reverse_twice is a regular Haskell function that takes a list of Int and returns a Boolean, based on the validity of what is being proposed—specifically, that two reverses cancel each other out. Here is the neat part: prop_reverse_twice is *also* a domain-specific statement and as such can be considered a sublanguage inside Haskell. This style of using functions (in this case, functions with names prefixed with prop_, taking a number of typed arguments, and returning a conditional) is a small language. The property written in Haskell is also an EDSL for properties, called QuickCheck.[4] This EDSL can be run using a function also called quickCheck:

```
GHCi>quickCheck prop_reverse
_twice
+++ OK, passed 100 tests.
```

By running quickCheck with this explicit and specific property, the EDSL executes inside Haskell. The quickCheck function generates 100 test cases for the property and executes them on the fly. If they all hold, then the system prints a message reflecting this. The test cases are generated using the type class system—QuickCheck gives specific types the power of test-case generation—and the quickCheck function uses this to generate random tests.

As an example of an incorrect property, consider this property for reverse.

```
prop_reverse (xs::[Int]) ys =
   reverse xs ++ reverse ys
      == reverse (xs ++ ys)
```

This states that the reverse of two distinct lists is the same as the reverse of both lists appended together, but this property is false.

```
GHCi>quickCheck prop_reverse
Falsifiable, after 5 tests:
 [0]
 [2,-2]
```

It turns out that this sort of minilanguage is really useful in practice. Despite the simplicity of how Haskell is being used, the QuickCheck EDSL provides a way of thinking about and directly expressing properties. It has additional functionality, including the ability to generate random function arguments, to control the distribution of the random test cases, and to state preconditions of a property. From this DSL, many other implementations of these ideas have been constructed. There is even a Swedish company, QuviQ, that sells a QuickCheck for the concurrent programming language Erlang.

Example EDSL: Kansas Lava. To take another example, consider describing hardware. Hardware description languages and functional languages have long enjoyed a fruitful partnership. Lava is the name given to a class of Haskell DSLs that implement a function-based version of the hardware description language Ruby.[12,13] Not to be confused with the modern programming language of the same name, Ruby was based on relations, which was in turn inspired by the seminal work in μFP.[21]

Kansas Lava[11] is a Haskell-hosted DSL that follows the Lava line of research. It is a language for expressing gate-level circuits. Haskell abstractions allow the programmer to work at a slightly higher level of abstraction, where the model is one of recursive components communicating via synchronized streams. Kansas Lava has been used for the generation of high-performance circuits for telemetry decoders, though the model used is general.

As an example of Kansas Lava, consider Figure 2: This circuit connects two multiplexers (mux2), an adder, and a register to give a circuit that counts the number of clocked pulses on a signal inc. The circuit takes two

Figure 3. Schematic Kansas Lava parity counter.

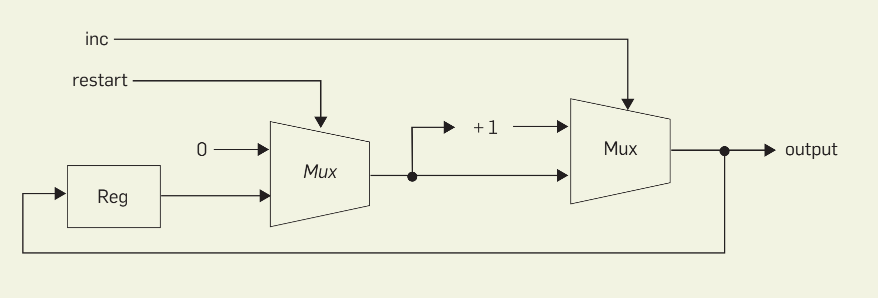
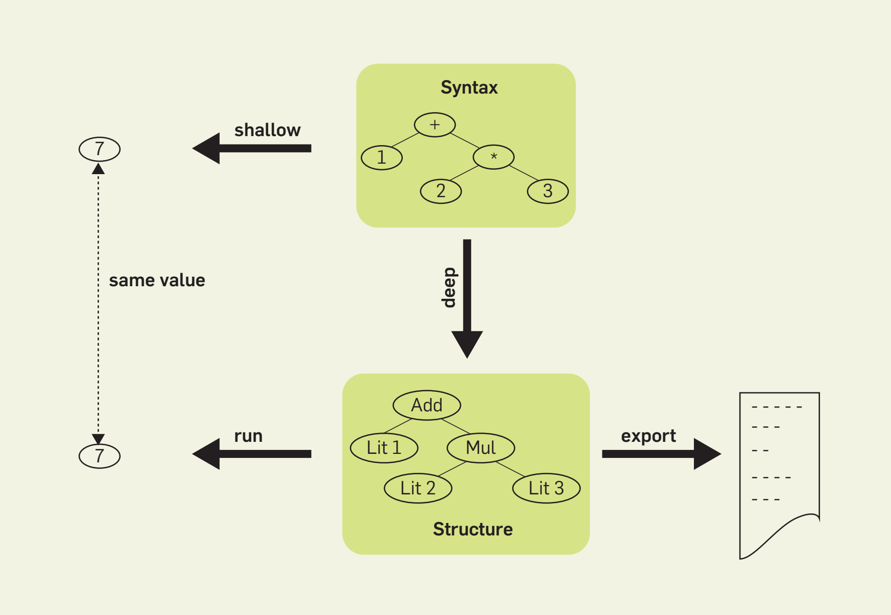


Figure 4. Running Kansas Lava as a simulation.

```
GHCi> toSeq (cycle [True,False,False])
True : False : False : True : False : False : True : False : False : ...
GHCi> counter low (toSeq (cycle [True,False,False]))
1 : 1 : 1 : 2 : 2 : 2 : 3 : 3 : 3 : ...
```

Figure 5. Shallow and deep embedding of arithmetic.



clocked signals and returns a clocked signal that explicitly operates using the same clock, because they share the same type. The use of arithmetic is understated, but simply uses (via overloading) the standard syntax for addition; the `Num` constraint allows this. Figure 3 illustrates the circuit intended for this description.

You can simulate sequential circuits with the same directness as the combinational functions invoked (see Figure 4).

As well as basic signal types, you can build circuits that operate on Haskell functions directly, provided the domain of the function is finite. The `Rep` capability is used to signify that you can enumerate all possible representable values in a type, giving the `funMap` function:

```
funMap :: (Rep a, Rep b)
     => (a -> Maybe b)
     -> Signal a
     -> Signal b
```

The generated circuit is implemented using a ROM, and you can generate control logic directly in terms of Haskell functions and data structures. As an example, consider a small ROM that stores the square of a value:

```
squareROM :: (Num a, Rep a)
 => Signal a -> Signal a
squareROM = funMap
    (\ x -> return (x * x))
```

In this way, direct Haskell functions can be lifted into the `Signal` world. Notice how the `squareROM` function is not specific about size but is completely generic, requiring only the type of the argument stream to be representable as a number.

The clock-squaring ROM can now be used at specific types. For example, at eight-bit you can generate the following:

```
GHCi> squareROM (toSeq [0,1..] ::
 Signal Word8)
 0 : 1 : 4 : 9 : 16 : 25 : 36 : 49 : 64 :
 81 : 100 : 121 : 144 : 169 : 196 :
 225 : 0 : ...
```

This level of circuit specification has been used to great effect in many Lava and Lava-like languages. One notable instance is Hawk,[15] a Lava-like EDSL that was used to specify the entire micro-architecture of the Pentium Pro, including the super-scaler design, and register bypass capabilities.

Now, if DSLs are so powerful as an idiom for library design, then why have they not taken over? As a means for expressing things that can be *simulated*, EDSLs are an invaluable design pattern; however, not everything is a simulation. What if you wanted to use an EDSL to express something you want to run somewhere else, not inside the Haskell system? Can Lava be used to generate circuits to be run on FPGAs (field-programmable gate arrays)? Can EDSLs be used to generate code for embedded processors or GPUs? Such an ability—to synthesize external solutions—would be extremely useful. The EDSL idiom can be extended to do so, with significant caveats. The remainder of this article is about how to capture and offshore work from inside an EDSL, what this capability can be used for, and what the limitations are.

Deeply Embedded Domain-Specific Languages

EDSLs are simply a way of thinking about a library of provided functions, often called *combinators*, because they combine their arguments into terms inside the DSL. In the previous Lava example, the `register` combinator takes an initial value—and an incoming stream of values—and provides the new stream, delayed by a single cycle, with the initial value occupying the initial cycle. Critically, `register` is compositional; it combines smaller parts of the DSL to make larger solutions. If a DSL follows this composability carefully by design, an important alternative implementation is possible.

The most common flavor of EDSL is one that uses so-called shallow embedding, as seen in Figure 1(2a), where values are computed directly. The result of a computation in a shallow EDSL is a value. All the examples so far are shallow. There is another class of EDSLs, however: specifically, those that use a deep embedding to build an abstract syntax tree, as shown in Figure 1(2b). The result of a computation inside a deeply embedded DSL (deep EDSL) is a structure, not a value, and this structure can be used to compute a value or be cross-compiled before being evaluated.[7] Such deep EDSLs follow the composability mantra pedantically, by design and mandate.

Historically, EDSLs have been shallow—simply a way of structuring an API for a library. Deep EDSLs, however, have the ability to *stage* code—that is, executing a program can generate another program, much like the well-known YACC DSL, but at the cost of significantly restricting what forms of the DSL can generate valid output.

There are a growing number of deep EDSLs, along with a body of research around their form and limitations. The unifying theme is that deep EDSLs can be pragmatic, productive, and useful.

This section investigates the basic structure of a deep EDSL compared with a shallow EDSL, and looks at three pragmatic tricks for improving the usefulness of deep EDSLs.

Building a deep EDSL. A deeply embedded DSL exposes its own composition and structure. Rather than using functions operating directly on values (a shallow DSL), a deep DSL builds a structure, then allows some secondary agent to provide interpretation of this structure. To make this idea concrete, consider a DSL for arithmetic, with addition, subtraction, multiplication, and constants. For a shallow embedding, running this DSL is trivial; you just use the built-in arithmetic. A deep embedding is where things get interesting. Consider a data type for our arithmetic:

```
data Expr where
  Lit :: Integer -> Expr
  Add :: Expr -> Expr -> Expr
  Sub :: Expr -> Expr -> Expr
  Mul :: Expr -> Expr -> Expr
  deriving Eq
```

Now overload the arithmetic to use this Expr data type; in Haskell, Num is the overloading for integral arithmetic:

```
instance Num Expr where
  fromInteger n = Lit n
  e1 + e2 = Add e1 e2
  e1 - e2 = Sub e1 e2
  e1 * e2 = Mul e1 e2
```

By building expressions of type Expr, you can observe the structure of the computation:

```
GHCi> 1 + 2 * 3 :: Expr
Add (Lit 1) (Mul (Lit 2) (Lit 3))
```

This is profound, and it is the key idea that makes deep embeddings work. You can write an expression and extract a tree of *what* to do, not a direct result. With deep embeddings, it is common also to write a run function that computes the result of a captured computation:

```
run :: Expr -> Integer
run (Lit n) = n
run (Add a b) = run a + run b
run (Sub a b) = run a - run b
run (Mul a b) = run a * run b
```

Figure 5 illustrates the differences between shallow and deep DSLs, and how a deep embedding combined with a specific run function gives the same result. For a deep embedded DSL, the run function restores the capability of the shallow embedding, but another function takes the embedded structure and uses it in some creative way.

To make deep DSLs practical, there are two additional tricks in the DSL folklore that are almost always used. The first trick allows the capture of functions, via dummy arguments. The second trick can observe loops, via some form of observable sharing.

How to extract a deep embedding from a function. Expressing function calls in terms of constructors and building expression trees is useful, but by itself is a gimmick. With careful construction, however, you can also capture function definitions, as well as other syntactical structures, directly from a deep embedding. It is at this point that the idea of capturing code, then using the captured code to execute code on a different target, becomes possible. Consider a simple function to add one to its argument:

```
f :: Expr -> Expr
f x = x + 1
```

Here is a function that acts over the new type Expr and returns a new Expr. How can you capture this function? The trick is to invent a unique Expr and pass it as a (dummy) argument to f:

```
data Expr where
  Lit :: Integer -> Expr
  Add :: Expr -> Expr -> Expr
  Sub :: Expr -> Expr -> Expr
  Mul :: Expr -> Expr -> Expr
  —new constructor Var
  Var:: String -> Expr
```

You can now run the function directly and see the result in your deep embedding, or pass in the Var argument and see the actual function in Figure 6: This is remarkable! You have

Figure 6. Capturing the structure of a function.

```
-- Just running the function
GHCi> f 4
Add (Lit 4) (Lit 1)
-- reifying the function, using our unique Var.
GHCi> f (Var “x”)
Add (Var “x”) (Lit 1) -- reified version of the function
```

Figure 7. A deep embedding for Lava.

```
data Signal where
  Register :: a -> Signal a                         -> Signal a
  Mux2     :: Signal Bool -> (Signal a,Signal a) -> Signal a
  Lit      :: a                                     -> Signal a
  Add      :: Signal a -> Signal a                  -> Signal a
  --the Var trick
  Var      :: String a                              -> Signal a

instance Num a => Num (Signal a) where
  a + b = Add a b

mux2 :: Signal Bool -> (Signal a,Signal a) -> Signal a
mux2 c (a,b) = Mux2 c (a,b)

register :: a -> Signal a -> Signal a
register d s = Register d s

--Now, when attempting to extract counter, things go horribly wrong:

GHCi> counter (Var “restart”) (Var “inc”)
Mux2 (Var “inc”) (Add (Mux2 (Var “restart”) (Lit 0,Register 0 (Mux2 (Var
“inc”) ...
```

run a function with a dummy argument (called the *prototypical argument*) and extracted the body of the function.

This idea scales to multi-argument functions. Consider g:

```
g :: Expr -> Expr -> Expr
g x y = x * x + y + 2
```

Two prototypical arguments to g will capture the function:

```
GHCi> g (Var “x”) (Var “y”)
Add (Add (Mul (Var “x”) (Var
“x”)) (Var “y”)) (Lit 2)
```

There are many places this design pattern can be used. One example is the specification of surface textures as functions; it is possible to export these into code executable on GPUs, simultaneously lifting the abstractions used to write textures and speeding up how fast an implementation of the same operations would run. There is nothing that is specific about Haskell or even functional languages here. Indeed, the same ideas have been used in Java for a VHSIC Hardware Description Language (VHDL) generator.[2] Haskell, with its powerful abstractions, allows deep DSLs almost to feel like a straightforward shallow embedding.

How to spot a loop. Lava programs are written as equations of recursive bindings. An attempt to build a deep embedding of Lava directly will lead to an infinite cycle of structures. To illustrate the challenge, let's build a deep embedding of Lava, see where it goes wrong, and fix it using a technique called observable sharing.

First the Lava language needs a structure. In Figure 7, we define the functions used before but give them a deep embedding, called Signal.

The output tree is infinite. What has happened is the recursive definitions are unrolling during attempts to reify the function, or more specifically, the body of counter is looping. At this point, the EDSL community was stymied. There were efforts to use monadic structure, where the loop was expressed using do-notation,[8] making the loop an observable effect. There was an unsafe extension to observe a limited form of sharing by circumventing part of the purity of Haskell, called observable sharing.[5] There was also an extension of the Haskell I/O mechanism that allowed loops to be observed indirectly, called I/O-based observable sharing.[9] The net effect of all three mechanisms is that the observed tree is rendered as a graph with named edges.

At this point Haskell rescues us from some complexity. Advanced type-system mechanisms, such as higher-kinded arguments, allow a structure to be either a tree or graph, depending on type-level instantiation. Omitting the details here, the reified function is a tree with sharing, then translated into a graph with explicit sharing. The final result for this example entity counter is:

```
GHCi> reify (counter
 Var “restart”) (Var “inc”))
[(0,MUX2 1 (2,3)),
 (1,VAR “inc”),
 (2,ADD 3 4),
 (3,MUX2 5 (6,7)),
 (4,LIT 1),
 (5,VAR “restart”),
 (6,LIT 0),
 (7,REGISTER 0 0)]
```

In this output, each uppercase constructor corresponds to its deep-embedding constructor. A quick inspection shows that the circuit has been captured, as shown in Figure 3. From this netlist-style structure, generating VHDL is straightforward. For the example of four-bit numbers, the VHDL is provided in Figure 8.

These two tricks (prototypical argument and I/O-based observable sharing) are the technical fundamentals of Kansas Lava. On top of this base, and with help from the Haskell type system, an entire ecosystem for circuit generation has been developed. The DSL idiom allows programmers to use high-level abstraction in Haskell and generate efficient circuits. Not all is rosy, however; writing a Lava program is not the same as writing a Haskell program because of the limitations of deep embeddings.

Figure 8. VHDL generated by Kansas Lava for counter.

```
entity counter is
  port(rst : in std_logic;
       clk : in std_logic;
       clk_en : in std_logic;
       restart : in std_logic;
       inc : in std_logic;
       output : out std_logic_vector(3 downto 0));
end entity counter;
architecture str of counter is
  signal sig_2_o0 : std_logic_vector(3 downto 0);
  ...
begin
  sig_2_o0 <= sig_5_o0 when (inc = ‘1’)  else sig_6_o0;
  sig_5_o0 <= std_logic_vector(...);
  sig_6_o0 <= “0000” when (restart = ‘1’) else sig_10_o0;
  sig_10_o0_next <= sig_2_o0;
  proc14 : process(rst,clk) is
  begin
    if rst = ‘1’ then
      sig_10_o0 <= “0000”;
    elsif rising_edge(clk) then
      if (clk_en = ‘1’) then
        sig_10_o0 <= sig_10_o0_next;
  ....
end architecture;
```

A Deep Embedding Is Only Half a Program

The basis of a deep EDSL is one of constructiveness. Functional programming is about constructing *and deconstructing* values. Because of this, a deep embedding cannot reify any pattern matching—or even direct usage of if-then-else—and other control flow. Kansas Lava sidestepped this—for example, by using a mux2 constructor, which encodes choice. How much further can the idiom be pushed if you need to be deconstructive? The result is surprising. Let's start with the three capabilities:

- Basic expressions can be captured by constructing a tree that is an analogue to your syntax.
- Functions can be captured using a fake unique argument.

- Local bindings can be observed using some form of observable sharing.

With these three comes an automatic fourth capability:

- The host language provides a built-in macro capability to the embedded language. Any part of Haskell (including control flow and pattern matching) can be used to *generate* the embedded language.

There are also extensions to the basic techniques. The principal ones are:

- Internal function calls can be captured as nodes on a graph, rather than directly inlined.[14] This helps compilation of large programs, giving a basic separate compilation capability.
- The `do` statement can be reified by normalization.[16,19,22] This result, called *monadic reification*, is surprising. There are strong technical reasons to believe monadic reification should be impossible; however, the normalization refactors the constraints that, by themselves, would be impossible to solve and matches them up, one-on-one, with a matching witness, allowing the whole `do`-notation to be solved and reified. Monadic reification is a recent discovery but has already been used in several deep DSLs, including Feldspar[1] and Sunroof.[3]
- Control flow is problematic and cannot be used directly, but there is a generalization of Haskell Boolean that does allow deep-embedding capture.[6] Using this library, a DSL with control flow can be constructed, but it needs to be explicit code, at the DSL level, using constructors. The `mux2` function used previously is a simplification of this idea. The usage is clumsy but workable, and we should be able to do better.

Where does this leave deep DSLs? They are clearly a useful design pattern for the language implementer, but they come with costs and limitations. How can we therefore push the state of the art and allow more of the Haskell language to be reified? There are two primary shortcomings. One we have discussed already: control flow and pattern matching remain a thorn in deep DSLs.

Parametric polymorphism, one of the strengths of a functional program, is the other issue for deep DSLs. A specific structure is needed to represent what has been captured, and arbitrary polymorphism interferes with this. Current systems sidestep this issue by always instantiating at a specific type, but this is expensive because the size of the captured program can expand exponentially. Polymorphism was the technical reason it was thought that monadic reification was not possible, but in that case it was sidestepped by normalization; this technique does not generalize to all polymorphism.

A deep DSL is a value-level way of extracting an expression, but there are other ways. Quasi-quoting is a mechanism for extracting expressions, but at the syntactic level. Haskell comes with an extensive template system called Template Haskell[20], which is often used for DSLs. There is a sense of unease with such solutions; however, in much the same way the C preprocessor is used even though it is not considered elegant. The principal issue is that the syntax of Haskell is huge, consisting of around 100 syntactical terms. An expression-based solution, such as a deep embedding, can avoid the need to rewrite front translations. Quasi-quoting has one important advantage: specifically, it can cope with control flow and deconstruction of values. Perhaps the future of deep DSLs is some hybrid between expression generation and quasi-quoting, combining the best of both systems.

Acknowledgments

This article is based upon work supported by the National Science Foundation under Grant No. CCF-1117569, and was originally presented as a master class under the Scottish Informatics & Computer Science Alliance Visiting Fellow program in Nov. 2013. The Kansas Lava examples and description were adapted from an earlier article about Lava written by the author.[10]

Related articles
on queue.acm.org

OCaml for the Masses
Yaron Minsky
http://queue.acm.org/detail.cfm?id=2038036

The World According to LINQ
Erik Meijer
http://queue.acm.org/detail.cfm?id=2024658

DSL for the Uninitiated
Debasish Ghosh
http://queue.acm.org/detail.cfm?id=198975

References

1. Axelsson, E., Claessen, K., Sheeran, M., Svenningsson, J., Engdal, D. and Persson, A. The design and implementation of Feldspar: an embedded language for digital signal processing. In *Proceedings of the 22nd International Conference on Implementation and Application of Functional Languages*. Springer-Verlag, 2011, 121–136.
2. Bellows, P. and Hutchings, B. JHDL—An HDL for reconfigurable systems. *Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (1998).
3. Bracker, J. and Gill, A. Sunroof: A monadic DSL for generating JavaScript. *Practical Aspects of Declarative Languages.* M. Flatt and H-F Guo, eds. *Lecture Notes in Computer Science 8324.* Springer International Publishing, 2014, 65–80.
4. Claessen, K. and Hughes, J. Quickcheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming* (2000), 268–279.
5. Claessen, K. and Sands, D. Observable sharing for functional circuit description. In *Proceedings of Asian Computer Science Conference, Lecture Notes in Computer Science*. Springer Verlag, 1999.
6. Elliott, C. Boolean package; hackage.haskell.org.
7. Elliott, C. Finne, S. and de Moor, O. Compiling embedded languages. *Journal of Functional Programming 13*, 2 (2003).
8. Erkök, L. and Launchbury, J. Recursive monadic bindings. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming* (2000), 174–185.
9. Gill, A. Type-safe observable sharing in Haskell. In *Proceedings of the 2nd ACM SIGPLAN Haskell Symposium* (2009), 117–128.
10. Gill, A. Declarative FPGA circuit synthesis using Kansas Lava. *The International Conference on Engineering of Reconfigurable Systems and Algorithms* (2011).
11. Gill, A., Bull, T., Farmer, A., Kimmell, G. and Komp, E. Types and associated type families for hardware simulation and synthesis: the internals and externals of Kansas Lava. *Higher-Order and Symbolic Computation*, (2013), 1–20.
12. Hutton, G. The Ruby interpreter. *Research Report 72*, (1993). Chalmers University of Technology.
13. Jones, G. and Sheeran, M. Circuit design in Ruby. *Formal Methods for VLSI Design.* Jorgen Staunstrup, ed. Elsevier Science Publications, 1990.
14. Mainland, G. and Morrisett, G. Nikola: Embedding compiled GPU functions in Haskell. In *Proceedings of the 3rd ACM Haskell Symposium on Haskell*, (2010), 67–78.
15. Matthews, J., Cook, B. and Launchbury, J. Microprocessor specification in Hawk. In *Proceedings of the International Conference on Computer Languages* (1998), 90–101.
16. Persson, A., Axelsson, E. and Svenningsson, J. Generic monadic constructs for embedded languages. *Implementation and Application of Functional Languages*, 2012, 85–99. Springer.
17. Peyton Jones, S.L., ed. *Haskell 98 Language and Libraries—The Revised Report.* Cambridge University Press, 2003.
18. Peyton Jones, S.L. and Wadler, P. Imperative functional programming. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (1993), 71–84.
19. Sculthorpe, N., Bracker, J., Giorgidze, G. and Gill, A. The constrained-monad problem. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, (2013), 287–298.
20. Sheard, T. and Jones, S.P. Template metaprogramming for Haskell. *ACM SIGPLAN Haskell Workshop 02.* M.M.T. Chakravarty, ed. ACM Press, Oct 2002, 1–16.
21. Sheeran, M. μFP, a language for VLSI design. In *Proceedings of the ACM Symposium on LISP and Functional Programming*, (1984), 104–112.
22. Svenningsson, J. and Svensson, B.J. Simple and compositional reification of monadic embedded languages. In *Proceedings of the Int'l Conference on Functional Programming*, (2013), 299–304.

Andy Gill (andygill@ku.edu) is an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Kansas.

DOI:10.1145/2605176

Article development led by acmqueue
queue.acm.org

"Mostly functional" programming does not work.

BY ERIK MEIJER

The Curse of the Excluded Middle

THERE IS A trend in the software industry to sell "mostly functional" programming as the silver bullet for solving problems developers face with concurrency, parallelism (manycore), and, of course, big data. Contemporary imperative languages could continue the ongoing trend, embrace closures, and try to limit mutation and other side effects. Unfortunately, just as "mostly secure" does not work, "mostly functional" does not work either. Instead, developers should seriously consider a completely fundamentalist option as well: embrace pure lazy functional programming with all effects explicitly surfaced in the type system using monads.

Like dieters falling for magic 10-minute-miracle exercise gadgets, developers seem ready to fall for easy solutions to the latest crises in their field. Recently, many are touting "nearly functional programming" and "limited side effects" as the perfect weapons against the new elephants in the room: concurrency and parallelism. Few want to accept that the benefits necessarily come at the cost of the convenience of latent effects in common operations such as I/O, just as dieters would rather not admit that the benefits of exercise necessarily come at the cost of time and sweat.

Just like "mostly secure," "mostly pure" is wishful thinking. The slightest implicit imperative effect erases all the benefits of purity, just as a single bacterium can infect a sterile wound. On the other hand, radically eradicating all effects—explicit and implicit—renders programming languages useless. This is the curse of the excluded middle: you must confront effects seriously by either accepting that programming is ultimately about mutating state and other effects, but for pragmatic reasons tame effects as much as possible; or by abolishing all implicit imperative effects and making them fully explicit in the type system, but for pragmatic reasons allow occasional explicit effects to be suppressed.

The Problem

Imperative programs describe computations by repeatedly performing implicit effects on a shared global state. In a parallel/concurrent/distributed world, however, a single global state is an unacceptable bottleneck, so the foundational assumption of imperative programming that underpins most contemporary programming languages is starting to crumble. Contrary to popular belief, making state variables immutable comes nowhere close to eliminating unacceptable implicit imperative effects. Operations as ordinary as exceptions, threading, and I/O cause as much hardship as simple mutable state. Consider the following C# example (thanks to Gavin Bierman) that filters an array to retain all values between 20 and 30, as shown in Figure 1.

Because `Where` is lazy (or uses deferred execution), the effects in the predicates used in `q0` and `q1` are interleaved; hence evaluating `q1` prints all values between 20 and 30 as if the predicates were intersected:

```
1? Less than 30; 1? More Than
20; 25? Less than 30; 25? More
Than 20; [25];40? Less than 30;
5? Less than 30; 5? More Than
20; 23? Less than 30; 23? More
Than 20; [23];
```

The average programmer would surely expect `q0` to filter out all values above 30 before `q1` starts and removes all values smaller than 20, because that is the way the program was written, as evidenced by the semicolon between the two statements. Any cross-dependencies between the predicates come as a nasty surprise.

Here is another example: mixing laziness with exceptions. If an exception is thrown in the body of a closure passed to the `Select` (map) function, because of deferred execution the exception is not thrown inside the scope of the `try-catch` handler. Instead, the exception is thrown once evaluation is forced by the `foreach` loop, with no handler in sight, as shown in Figure 2.

Effects interfere with language features in other ways, such as the interplay between closures and the disposable pattern. The next example opens a file for reading and captures the file variable in a closure that escapes the lexical scope of the using block. In C# the `using` statement causes the variable initialized at the entry of the block to be automatically disposed when control-flow reaches the end of the block. Hence, by the time the closure is called, the file will have been disposed of, causing a surprising exception far away in time and space from where the exception-throwing code resides in the static source code (see Figure 3).

The lexical scope of `try-catch` and `using` blocks does not mix well with the dynamic extent of side-effecting closures.

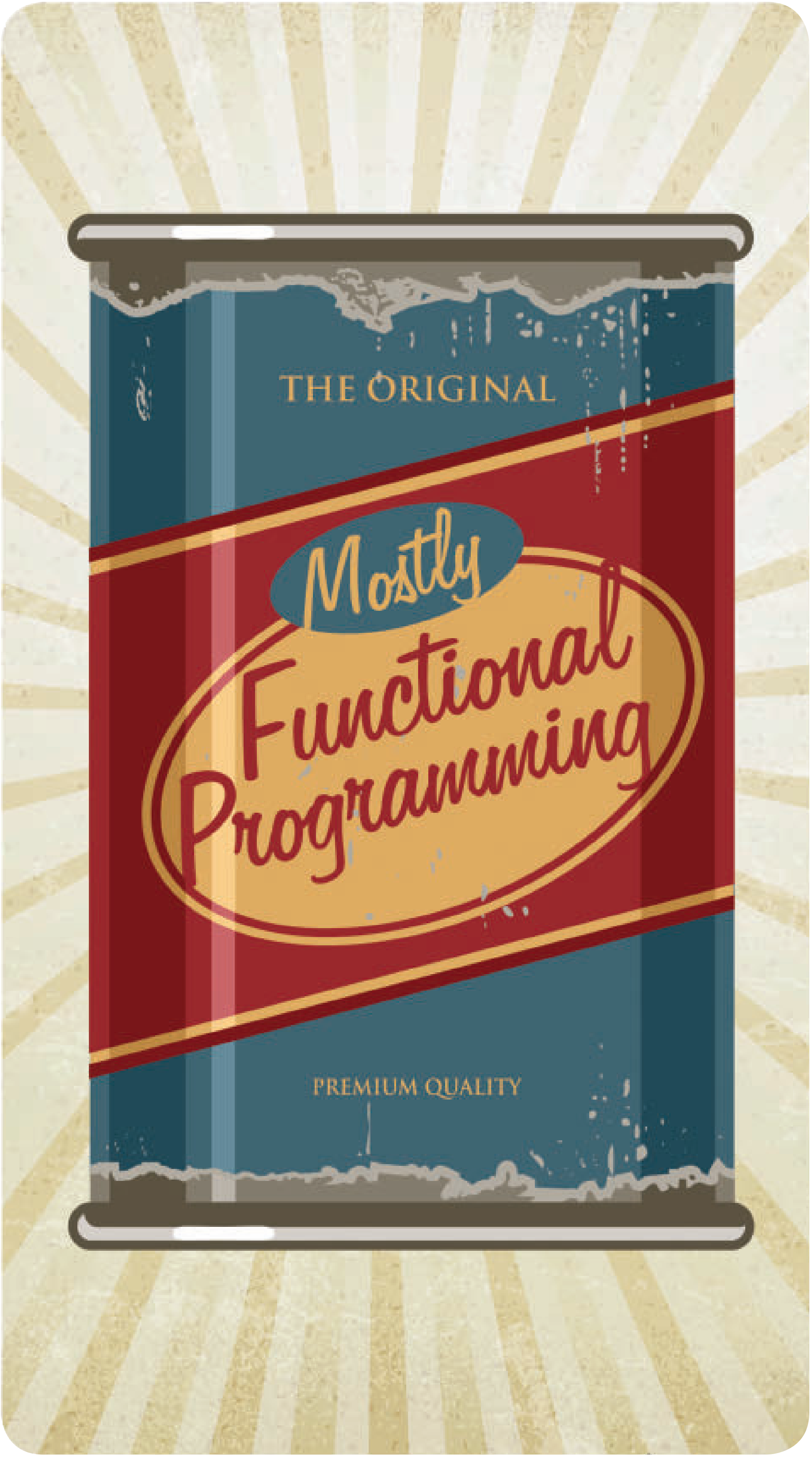


ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

Down the Rabbit Hole

If the examples with laziness and closures look farfetched, let's look at a method that in printf debugging style traces its return value to the console. As innocent as this may look, it is now unsafe for a compiler to perform optimizations as simple as common-subexpression elimination:

```
string Ha() { var ha = "Ha";
Console.Write(ha); return ha; }

// prints HaHa
var haha = Ha()+Ha();

// prints Ha
var ha = Ha();
var haha = ha+ha;
```

But wait—it is actually much worse. Simply creating a new object is an observable side effect. Even when called with the same arguments, a constructor will return a different result each time, as can be observed via the GetHashCode or ReferenceEquals method, among others:

```
var a = new Object();
//58225482, for instance
Console.WriteLine(a.GetHash-
Code());

var b = new Object();
// 54267293, for instance
Console.WriteLine(b.GetHash-
Code());

Debug.Assert(a != b);
```

To make constructors pure, you must insist on value semantics for all objects, which implies the elimination of all shared mutable state in objects. Now the essence of object-oriented programming is lost: encapsulation of state and behavior in a single unit.

These examples illustrate another harmful consequence of implicit imperative effects: it forecloses many common optimizing transformations. Since side effects implicitly alter the program's global environment, it is usually unfeasible to isolate and localize effects. As a result, imperative programs are mostly noncompositional, making it difficult for both programmers and compilers to reason about them.

Bad as this is, surely it is enough just to abolish state mutation to make the code pure. No! Unfortunately, it is not enough simply to make all fields in a class `readonly` and disallow assignments to local variables in order to tame the state monster. The Cω program here shows that threads can easily simulate state, even though there are no assignments or mutable variables anywhere to be seen in the code. As shown in Figure 4, the private channel Value(T current), carries the state of the cell, piggybacking on the hidden mutable state of the thread's message queue within the cell.

Passing state around over a private channel is embraced as the *foundation* of active objects or actors in Erlang. Figure 5 illustrates the Erlang version of the mutable `Cell` program above, using a tail-recursive function `cell(Value)` to hold the state of the `Cell`.

Note how this Erlang actor basically encodes an object with dynamic method dispatch using the pattern-matching, message-sending, and recursion primitives of the language, which you may happily leverage to implement mutable references, sabotaging the fact that the Erlang language does not natively expose mutable state.

These examples are just the tip of the iceberg. The fundamental problem with side effects is there are many uncontrollable ways to observe them, and even worse, it is often possible to simulate one effect with another. Adding recursion to pure combinatorial circuits makes it possible to build flip-flops that provide mutable state. You can prove that state and delimited continuations suffice to simulate any effect,[1] and that unchecked exceptions can simulate continuations.[4] You cannot be careful enough in dealing with effects.

The examples just presented show how tricky and subtle side effects can be, popping up where they are least expected. Maybe, then, you might ask, "With care and discipline can you transform an imperative language into a pure one by avoiding features that cause problems?" To do this, you would have to remove *all* of its intrinsic effects. Even evaluation is an effect, and you must give up control of it to the compiler and runtime. The consequence is that such ultra-pure programs cannot interact with the user or the environment, cannot perform network I/O, react to user-interface events, read data from files, get the time of day, or generate random numbers. This is a terrible dilemma: If pure programs cannot leave any trace of ever being executed, then how can they be useful?

Though the situation seems bleak, fortunately there are several ways out of the pit. They all involve tastefully adding features instead of blindly removing them, as John Hughes observed in 1984 in his seminal paper, "Why Functional Programming Matters."[2]

Fundamentalist Functional Programming

Pure functional programming is programming with mathematical functions. This means the only way to express dependencies among values is by applying functions to arguments and harvesting values returned. Calling a function with the same arguments will return the same result every time. There is no way to keep a secret, hide a value in a little place to be picked up later, directly say do this before that, spin up a thread, throw an unchecked exception, or just print something to the console. This may seem rigid and fundamentalist. It is. But it is also powerful and enabling.

To understand how fundamentalist functional programming might help solve the concurrency problem, it is important to understand it is not just imperative programming without side effects, which, as we have seen, is useless. Rather, it harnesses the fundamentalist language of mathematical functions to express and encapsulate rich effects compositionally using monads. To build an intuitive model of monads, let's look at perhaps the simplest possible generic method:

```
T Identity<T>(T me) { … }
```

The type signature of this method claims that *for all* types `T`, given an argument of type `T`, the method will return a value of type `T`. That is not the whole truth; the imperative type signature is *insouciant*, permissive, too loose. It expresses only an approximation of the actual *mathematical* type of the method. It does not account for any side effects that might be hiding in its execution, such as the fact that it eagerly evaluates its argument; it might inspect the generic type parameter (so it does not work uniformly with all types T); it might throw exceptions, perform I/O, spin up threads, allocate new objects in the heap, and take a lock; it can access this, and so on.

The trick of pure fundamentalist functional languages is to have obsessive-compulsive types and expose all effects explicitly in the type signatures using monads, distinguishing between the types of pure values and the types of computations on values that may entail effects.

Informal Introduction to Monads

For a value of type a, let's write an effectful computation that can deliver a value of this type in the notation of a function application (M a), the Haskell syntax for generic types. One way to envision this effectful computation is with a machine for producing output values of type a, while explicitly denoting the effect M caused by the computation of the output value. Think of a factory in the real world that needs electricity and pollutes the environment as a side effect of producing goods. Note that a value of type M a is just a promise to produce a value of type a, and no effect is yet performed. To actually do something with a monadic value, there are two standard combinators that work on such effectful computations:

- The *infix application function* (ma>>=\a->f(a)), commonly called *bind*, executes the computation ma to expose its effects, calling the resulting value a, and passes that to function f. Evidently, the result of this application (f a) should again be a potentially side-effecting computation; hence, the type signature for bind looks like this:

```
(>>=):: M a -> (a -> M b) -> M b.
```

- The *injection function* (return a) injects a pure value into a computation. Its type signature is: return :: a -> M a.

In practice, each effect usually comes with so-called *nonproper morphisms*. These domain-specific operations are unique to that effect (see examples later in this article).

You can now abstract and formalize all effectful computations of the form

Figure 1. Conflating lazily evaluated selections.

```
static bool LessThanThirty(int x) {
  Console.Write(“{0}? Less than 30;”, x); return x < 30;
}
static bool MoreThanTwenty(int x) {
  Console.Write(“{0}? More than 20;”, x); return x > 20;
}

var q0 = new[]{ 1, 25, 40, 5, 23 }.Where(LessThanThirty);
var q1 = q1.Where(MoreThanTwenty);
foreach (var r in q1){ Console.WriteLine(“[{0}];”,r); }
```

Figure 2. Trying to catch a division by zero exception.

```
var xs = new[]{ 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };
IEnumerable<int> q;
try { q = xs.Select(x=>1/x); } catch { q = new int[]; }
foreach(var z in q){ Console.WriteLine(z): // throws here }
```

Figure 3. Using a closed file.

```
Func<string> GetContents;
using(var file = FileSystem.OpenTextFileReader(@”my file”)) {
   GetContents = ()=>file.ReadToEnd();
}
Console.WriteLine(GetContents()); // surprise! an exception
```

Figure 4. Joins + Threading = Mutable state.

```
class Cell<T> {
   Cell<T>(T init){ Value(init); }
   T Get() & async Value(T current){ return current; }
   async Set(T @new) & async Value(T old){ Value(@new); }
}
```

You can define a mutable Point class using readonly fields of type Cell:

```
class Point {
   readonly Cell<Int> x = new Cell<int>(0);
   readonly Cell<int> y = new Cell<int>(0);
}
```

Figure 5. Tail recursion and Threading = Mutable state.

```
new_cell(X) -> spawn(fun() -> cell(X) end).
cell(Value) ->
   receive
      {set, NewValue} -> cell(NewValue);
      {get, Pid}      -> Pid!{return, Value}, cell(Value);
      {dispose}       -> {}
   end.
set_cell(Cell, NewValue) -> Cell!{set, NewValue}.
get_cell(Cell) ->
   Cell!{get, self()},
   receive
      {return, Value} -> Value
   end.
dispose_cell(Cell) -> Cell!{dispose}.
```

above as an algebraic structure called a *monad* that supports the two generic operations for creating and propagating effects, `return` and `bind (>>=)`:

```
class Monad m where {
    (>>=)  :: m a -> (a -> m b) -> m b
   return :: a               -> m a
}
```

The mother of all monads in Haskell is the I/O monad, which represents all computations that have a global effect. As such, it comes with a large number of nonproper morphisms, most of which look familiar to imperative programmers, such as operations to read and write to the console, spinning up threads, and throwing exceptions. Once effects are explicit in, say, the I/O monad, operations such as allocating mutable variables, and reading and writing to them, must be lifted into the I/O monad (see Figure 6).

From the type of `forkIO`, you can immediately see that spinning up a thread is a side-effecting operation, which ensures any encoding of mutable cells using threads will be in the I/O monad as well. This prevents programmers from believing they are defining immutable types where they are not. Note that the I/O monad does not prevent the global state to be updated simultaneously from several threads, as the type of `forkIO` shows.

The real power of monads comes from the fact that computations are themselves just values that can be passed around as first-class citizens within the pure host language. This enables programmers to write new abstractions (domain-specific and custom-control structures) using the monadic primitives and nonproper morphisms.

For example, you can easily define a function that takes a list of side-effecting computations and executes each, collecting the results in a list as shown in Figure 7.

Figure 6. The penalty box of effects.

```
data IO a -- no need to know how this is implemented

putChar :: Char -> IO ()
getChar :: IO Char

newIORef   :: a            -> IO (IORef a)
readIORef  :: IORef a      -> IO a
writeIORef :: IORef a -> a -> IO ()

forkIO :: IO a -> IO ThreadID
```

Figure 7. Haskell, the world's finest imperative language.

```
sequence          :: Monad m => [m a] -> m [a]
sequence []       = return []
sequence (ma:mas) = ma >>= (\a -> sequence mas >>= (\as -> a:as))
```

Figure 8. Isolated transactions.

```
newTVar   :: a           -> STM (TVar a)
readTVar  :: TVar a      -> STM a
writeTVar :: TVar a -> a -> STM ()
```

The `retry` operation aborts the current transaction and waits until any transactional variables are changed, or it attempts to run another transactional block when chained via the `orElse` operator.

```
retry  :: STM a
orElse :: STM a -> STM a -> STM a
```

Finally, the function `atomic` injects the transaction monad into the general I/O monad. From that point on, no more transactional operations can be performed on the value.

```
atomic :: STM a -> IO a
```

A pure fundamentalist functional language is a very convenient host language for internal DSLs since it is just executable denotational semantics. As such, pure functional languages come a long way toward fulfilling the vision of P.J. Landin's influential paper published in 1966, "The Next 700 Programming Languages."[3]

The additional beauty of making effects explicit in the type system is that you can distinguish among various effects and freely introduce new ones. For example, you can define the monad `(STM a)` of transactional memory.[6] It contains nonproper morphisms for allocating, reading, and writing transactional variables as illustrated in Figure 8.

Since there is no (implicit) leakage from `I/O` to `STM`, the `STM` monad captures all the mechanisms needed to perform transactions without having to worry about rolling back arbitrary effects, since ordinary Haskell computations are assumed to be pure.

Side effects are not only difficult to tame, but they also are often sneaking in the back door. Even in the supposedly pure Haskell, there is a seemingly inconspicuous function called `unsafePerformIO :: IO a->a` that tells the compiler to "forget" about the side effects involved in evaluating its argument. This "almost function" is supposed to facilitate encapsulating benevolent side effects in an otherwise pure computation; `unsafePerformIO`, however, opens a Pandora's box because it subverts the Haskell type system, wrecking language guarantees and allowing any type to be cast to any other type:

```
unsafeCast :: a -> b
unsafeCast x = unsafePerformIO
(do{
    writeIORef castref x; readIORef castref
})
castref = unsafePerformIO (do{
newIORef undefined })
```

These examples show some of the power of treating side-effecting computations as values using monads, but the question remains unanswered if the average developer can deal with them. We believe the answer is a resounding yes, as confirmed by the fact that monads are also just what make LINQ tick. The LINQ Standard Query Operators are essentially monadic operators. For example, the `SelectMany` extension method directly corresponds to the Haskell bind `(>>=)` introduced earlier:

```
(>>=) :: m a -> (a -> m b) -> m b
M<T> SelectMany(this M<S> src,
Func<S, M<T>> f)
```

A major difference between polymorphism in Haskell and generics in Common Language Runtime (CLR), Java, or other object-oriented languages is that the Haskell definition of monads relies on higher-kinded polymorphism—that is, the monad class (interface) is parameterized by a type constructor instead of by a type. Usually, generics allow parameterization only over types—for example, `List<T>` and `Array<T>`—but do not allow parameterization over the container type `M` as in `M<T>` and then instantiate `M` with `List` or `Array`. As a consequence, the LINQ standard sequence operators that encode monads must rely on syntactical patterns, essentially overloads, instead of proper generics.

The ease with which you can define reusable abstractions over effects is why people often call Haskell the world's best imperative language. Maybe the real crisis the industry now faces is much worse than the perceived pain of adopting pure fundamentalist functional programming. The enthusiastic reception of LINQ is an indication that people are ready for change.

Alternatives

Common belief in the fundamentalist approach to programming is too much of a paradigm shift for ordinary programmers, and the way forward is to make existing imperative languages more pure by taming effects. The advocated approach is to tame effects in the dual way: assume all methods have ambient effects (are in the I/O monad), except for those marked with special modifiers such as `threadsafe`, `pure`, `immutable`, `readonly`, distinguishing between `val` and `var`, and so on, that signal the *absence* of events. While at first sight this may seem to be less obtrusive to the developer, this is arguably not the case since once in a pure context, you cannot call an impure function and hence must transitively annotate any method called from a pure method as well—just as adding an effect somewhere deep inside a pure expression in Haskell requires a complete refactoring to monadic style.

One problem with purity annotations is that they are not extensible—that is, users cannot define new "non-effects." As we have seen in the examples of monadic composition, it is essential to support users defining their effects and nonproper morphisms whenever necessary.

Second, purity annotations typically pertain to *functions*, whereas in Haskell, effects are not tied to functions, but to *values*. In Haskell, a function of type `f::A->IO B` is a pure function that given a value of type `A` returns a side-effecting computation, represented by a value of type `IO B`. Applying the function `f`, however, does not cause any immediate effects to happen. This is rather different from marking a function as being pure. As shown here, attaching effects to values enables programmers to define their own control structures that, for example, take lists of side-effecting computations into a side-effecting computation that computes a list.

There are many other proposals for reasoning about effects in imperative languages, such as linear or uniqueness types, ownership types, or, most recently, separation logic.[5] These all require a high degree of sophistication from both users and tools, however. This is extremely heavy mathematical machinery, particularly when compared with the simple equational reasoning of monads and pure functional programming, and thus does not really simplify the lives of ordinary programmers. One should not need a Ph.D. in theoretical computer science to be able to hack and reason about code.

Taming effects to make an imperative language pure is as painful for developers as being fundamentalist and making a pure language imperative.

Conclusion

The idea of "mostly functional programming" is unfeasible. It is impossible to make imperative programming languages safer by only partially removing implicit side effects. Leaving one kind of effect is often enough to simulate the very effect you just tried to remove. On the other hand, allowing effects to be "forgotten" in a pure language also causes mayhem in its own way.

Unfortunately, there is no golden middle, and we are faced with a classic dichotomy: the curse of the excluded middle, which presents the choice of either (a) trying to tame effects using purity annotations, yet fully embracing the fact that your code is still fundamentally effectful; or (b) fully embracing purity by making all effects explicit in the type system and being pragmatic by introducing nonfunctions such as `unsafePerformIO`. The examples shown here are meant to convince language designers and developers to jump through the mirror and start looking more seriously at fundamentalist functional programming.

Related articles on queue.acm.org

A Conversation with Erik Meijer and Jose Blakeley
http://queue.acm.org/detail.cfm?id=1394137

Multitier Programming in Hop
Manuel Serrano and Gérard Berry
http://queue.acm.org/detail.cfm?id=2330089

FPGA Programming for the Masses
David Bacon, Rodric Rabbah and Sunil Shukla
http://queue.acm.org/detail.cfm?id=2443836

References

1. Filinski, A. Representing monads. In *Proceedings of the 21st ACM Symp. on Principles of Programming Languages* (1994). ACM Press, 446–457.
2. Hughes, J. Why functional programming matters. *Computer J. 32*, 2 (1989), 98–107.
3. Landin, P.J. The next 700 programming languages. *Commun. ACM 9*, 3 (Mar. 1966), 157–166.
4. Lillibridge, M. Unchecked exceptions can be strictly more powerful than call/cc. *Higher-Order and Symbolic Computation 12*, 1 (1999), 75–104.
5. O'Hearn, P.W. A primer on separation logic (and automatic program verification and analysis). *Software Safety and Security; Tools for Analysis and Verification. NATO Science for Peace and Security Series* 33 (2012), 286–318.
6. Oram, A. and Wilson, G. *Beautiful Code: Leading Programmers Explain How They Think.* O'Reilly Media, 2007.

Erik Meijer (emeijer@applied-duality.com) is the founder of Applied Duality, Inc. and professor of Cloud Programming at TU Delft. He is perhaps best known for his contributions to programming languages such as Haskell, C#, Visual Basic, and Hack, and his work on LINQ and the Rx Framework.

DOI:10.1145/2605685

Article development led by acmqueue
queue.acm.org

High-level DSLs for low-level programming.

BY BO JOEL SVENSSON, MARY SHEERAN, AND RYAN R. NEWTON

Design Exploration through Code-Generating DSLs

DOMAIN-SPECIFIC LANGUAGES (DSLS) make programs shorter and easier to write. They can be stand-alone—for example, LaTeX, Makefiles, and SQL—or they can be embedded in a host language. You might think that DSLs embedded in high-level languages would be abstract or mathematically oriented, far from the nitty-gritty of low-level programming. This is not the case. This article demonstrates how high-level embedded DSLs (EDSLs) really can ease low-level programming. There is no contradiction.

A gentle introduction to EDSLs can be found in the previous article in this series: "Domain-specific Languages and Code Synthesis Using Haskell," in which Andy Gill considers the pros and cons of implementing a deeply embedded DSL, compared with a stand-alone compiler. Here, the story continues, posing a slightly different question: If you need to produce high-performance, low-level code in a language such as C or CUDA, is it worthwhile to use (or create) a code-generating EDSL, or should you just buckle down and write the low-level code by hand? Our answer is the EDSL may well be a reasonable option.

In 2010, one of this article's authors was tasked with porting high-performance benchmarks to run on a new computer architecture. One of these benchmarks was a parallel sort designed to use vector instructions[6]—SSE (streaming SIMD extensions) to be precise. To accomplish vectorization, this sort bottoms out to a bitonic sorting network once the arrays reach a small size. Sorting networks such as these are directly vectorizable. Of course, benchmarks of this kind have no reputation for maintaining clean abstractions in the code. Indeed, this particular benchmark had hard-coded sorting networks with 16, 32, and 64 inputs, using many pages of repetitive and inscrutable code:

```
...
xmm10 = _mm_min_ps(xmm1, xmm6);
xmm11 = _mm_max_ps(xmm1, xmm6);
xmm12 = _mm_min_ps(xmm2, xmm5);
xmm13 = _mm_max_ps(xmm2, xmm5);
xmm14 = _mm_min_ps(xmm3, xmm4);
xmm15 = _mm_max_ps(xmm3, xmm4);
...
```

This representation of the program has obvious drawbacks. For one, how can it be ported to a new architecture with different vector instructions? It's impossible—you have to write a new version. Yet, a glance at the bitonic sorter Wikipedia entry (http://en.wikipedia.org/wiki/Bitonic_sorter) shows the algorithm is a simple and elegant recursive function. Why can't that essential structure be captured, re-

ILLUSTRATION BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

moving the repetition from the example? Repetition can at least be reduced using C preprocessor macros:

```
#define SORT16(a,b,c,d,e,f,g,h,i,j,
k,l,m,n,p,q,r,s,t,u,v,w,x,y,z,za) \
   i = _mm_min_ps(a,h); \
   j = _mm_max_ps(a,h); \
   ...
```

SORT16 can then be reused to define SORT32. Still, C preprocessor macros do *not* offer the necessary compile-time programmability. Rather, the best solution is to write a simple program *generator* script that prints the repetitive code here to a file as output.

The lesson here is that software engineers should not hesitate to write programs that generate programs. If you can recognize regularities in your programs, then program generation is probably an option. There are plenty of sophisticated technologies to help with this *metaprogramming* (MetaOCaml, Scheme macros, Template Haskell, C++ template metaprogramming, among others). Indeed, because the DSLs discussed here are embedded, their host programs are in fact program generators (that is, metaprograms). These DSLs, however, also impose additional structure that yields several benefits.

DSLs offer safety. The script alluded to before is very primitive; it emits each generated program directly as a string. An EDSL instead emits an abstract syntax tree for the DSL program, and with that you can do whatever you like (see Andy Gill's article on page 42 for more details). In fact, if structured properly, an EDSL used to generate code can offer safety guarantees in the generated code; a well-typed EDSL program should guarantee well-typed generated code. This is a great way to find errors early. For example, if the EDSL is embedded in Haskell, the Haskell type checker finds errors that would have led to broken low-level programs. Finding those errors by debugging the low-level generated code could be much slower—for example, in embedded systems where build and debug times are long.

You can go even further and restrict the DSL to particular idioms so as to gain extra safety properties. For example, you can guarantee only memory-safe generated programs, as in Galois Inc.'s C code generation in its Ivory DSL (http://smaccmpilot.org/languages/).

DSLs enable smarter compilers. These days, DSLs for high performance focus on parallelism, but the degree to which parallel implementation details are exposed to the user varies among DSLs. Some, such as SQL and very high-level array DSLs,[2–4,11,16,21,22] hide everything but abstract data transformations, which are naturally parallel. These compilers can achieve good performance by limiting the kinds of communication allowed between parallel code regions, enforcing structured data-processing patterns such as map and reduce, and removing features that make auto-parallelization difficult (for example, aliasing, pointers, and arbitrary control flow). By leveraging these restrictions, you can often apply radical code transformations. An example of this would be fusion, the removal of intermediate data structures (for example, replacing `map f (map g arr)` with `map (f ∘ g) arr`).

When this approach works well—that is, when the compiler can map the latent parallelism in the declarative specification onto the target architecture efficiently—it is an attractive option. Sometimes, though, experimentation is necessary to find a good parallel decomposition that matches the target, which might be a GPU or FPGA, for example. The user then wants not only fine control over the generated code, but also easy ways to change it. For example, Kansas Lava, the hardware description language described in the aforementioned article by Andy Gill, is expressly intended for design-space exploration, and case studies in forward error correctors show how this leads to high-performance designs. At a higher level of abstraction, the Bluespec (http://www.bluespec.com/high-level-synthesis-tools.html) behavioral language allows algorithm experts to generate high-performance FPGA accelerators. A major selling point is that it is quick and easy to do architectural exploration, particularly using highly parameterized DSL code.

These same benefits apply to low-level software domains, as well as hardware. In general, not all performance tweaking and tuning can be automated, which creates a need for DSLs that enable systematic user-guided performance exploration.

DSLs abstract code-generation tactics. How should we give the programmer control over the generated code, along with easy ways to vary it? Because EDSL programs perform arbitrary computation at program-generation time, they can also encapsulate reusable code-generation tactics, in the guise of regular functions and data types. Let's take a look at an example: deferred array representations,[1,5,7,10] which represent the ability to generate elements (in this, they are similar to the concepts of *generators* or *iterators* found in many languages, but the specifics are quite different), rather than elements already residing in memory.

To explain deferred arrays, let's start with a simple program using the `map` and `reduce` functions from before:

```
let arr2 = map f arr1
    sum = reduce (+) arr2
    prd = reduce (*) arr2
...
```

Does `arr2` require memory for its storage (and, therefore, memory traffic to read and write it)? In most traditional languages the answer is an unqualified yes. In high-level array DSLs—which may or may not use deferred arrays—fusion optimizations *may* eliminate the intermediate array. This is possible because in many of these DSLs,[2,4,11,16] arrays are immutable and the DSLs themselves may even be side-effect free, making fusion opportunities easy to recognize in the compiler. Thus, very often a `map` operation *can* fuse into a `reduce` operation, which means the (parallel or sequential) loop that is eventually created for `reduce` contains the `map`'d function inside that loop.

In the example, however, `arr2` is used twice. Most array DSLs will choose not to inline `arr2` if it means duplicating work, or they will use a heuristic based on a static cost estimate for `f` to decide whether work duplication is worth it to enable fusion. Explicitly deferred arrays, on the other hand, leave this decision up to the programmer. They represent an array in terms of the computation that generates it, and support a fusion-by-default approach. The `map` in the previous example would be duplicated and fused into each `reduce`, unless the user explicitly makes an array "real" in memory by applying a function called `force`. Moreover, deferred arrays gain an additional benefit in the embedded DSL context: they never suffer function-call overhead for the functions used to represent arrays, because they are inlined as the EDSL host program performs code generation.

Deferred-array data types is one example of encapsulating code-generation tactics with a nice API. They provide programmers with high-level array operations, while retaining control over memory usage. They can also help avoid bounds checks without risk, when composing known producer and consumer functions from the provided library (for example, `map` and `reduce`). Later, we describe a language that provides even more control, with different control-flow patterns encapsulated in different variants of deferred arrays (push and pull). Finally, the article shows how deferred arrays can be used to make operations such as `map` highly generic (for example, a `map` operation that, depending on the context in which it is applied, generates a sequential loop *or* a parallel loop, and, if parallel, can operate at one of multiple scales inside a hierarchical parallel architecture, generating very different code in each case).

DSLs can mix it up (design explora-

tion). That last point—that high-level data operators can generate different code based on the context in which they are used—moves closer to the goal of design exploration. In what other ways can a DSL allow programmers to explore different implementation strategies, while changing many fewer lines of code than if they had to convert the generated code manually? One way is to take advantage of program generation to build highly parameterized designs. In a hardware-synthesis language, such as Bluespec or Lava, one wants designs that are flexible and do not commit to a specific circuit area. Likewise, later in this article, you will see GPU programs that are parameterized over the number of GPU hardware threads to use, sometimes changing the generated program structure (injecting sequential loops) if not enough threads are provided. This just-in-time determination of program structure would be difficult to accomplish in handwritten code. With it, we might accept a *list* of GPU programs in the host program and provision the GPU to run all those computations continuously and simultaneously, with each program adapting to the number of threads it is allotted.

Finally, with increased design-exploration power, a common desire is to auto-tune by generating many variants of a program, testing them all, and selecting the winner. Several DSLs internally perform such an auto-tuning process,[8,14,15] though that is beyond the scope of this article. Instead, this article aims to help you weigh the pros and cons of embedded DSLs that generate low-level code. It does so by presenting one concrete example: Obsidian, an embedded DSL that generates CUDA for GPU programming. First, let's review the CUDA programming model, before describing Obsidian in the remainder of this article.

A CUDA Primer

CUDA is a C dialect for GPU programming that NVIDIA provides. (Readers with previous experience with CUDA can skip this section.)

GPUs began as hardware accelerators for generating computer graphics, but have become increasingly general purpose, driven both by graphics programmers who want more flexibility and by a new breed of programmer, motivated not by graphics but by a desire to accelerate the data-parallel parts of non-graphics applications.

How should we give the programmer control over the generated code, along with easy ways to vary it?

A typical computer system containing a CUDA-capable GPU is split into two parts: the *host*, which refers to the CPU and main memory of the computer; and the *device*, which consists of the GPU and its associated memory. The device usually comes in the form of a PCI Express card holding the GPU and memory. The GPU consists of a number of multiprocessors (MPs), each containing some number of functional units (*CUDA cores* in NVIDIA terminology). Each MP also contains local shared memory, which can be regarded as a programmer-managed cache. The CUDA cores within an MP can communicate via this shared memory.

Massive parallelism. A GPU is capable of managing a large number of threads in flight. It thrives when thousands of threads (doing mostly identical work) are launched together. It has a hierarchical structure managing all these threads. Onto each MP are launched several *blocks* of threads, each consisting of up to 1,024 threads. Threads within a block (and only within the block) can communicate using the shared memory of the MP. The collection of blocks launched onto the GPU is called the *grid*. A group of 32 consecutive threads within a block is a *warp*. Warps execute in lock-step; all threads perform the same instruction at any given moment. Threads within a warp that do not partake in the computation are turned off.

Scalable architecture. GPUs may contain as little as one or as many as 15 MPs. A CUDA program should be able to run on any of the GPUs along the scale. The CUDA programming model mirrors the GPU hierarchy. The programmer writes an SPMD (single-program multiple-data) program that is executed by the threads within a block. Many instances of this program also execute across all blocks launched. All instances are independent and can be launched in any order, or in parallel, across available MPs.

The upshot of all this is that programming a GPU really doesn't much resemble programming a multicore machine with two, four, or eight cores.

Programmers must launch thousands of threads and figure out how they should communicate within the constraints of the GPU. This means they need to think about warps, blocks, and grids, like it or not. (As will be seen, however, a DSL can still make it *easier* to map abstract data transforms onto the hierarchy.)

The references at the end of this article include links to further reading on CUDA programming and GPU architecture.[12,13]

The Obsidian Language

In CUDA, parallel `for` loops are implicit; the computation is described at the element level. Which elements to access (where to load and store data) is expressed as a function of a thread's identity—that is, threads must ask "where am I?" and answer the question by computing with `blockIdx`, `blockDim`, and `threadIdx`. Obsidian programs, in contrast, describe array-to-array computations at an aggregate level, replacing indexing arithmetic with collective operations. For example:

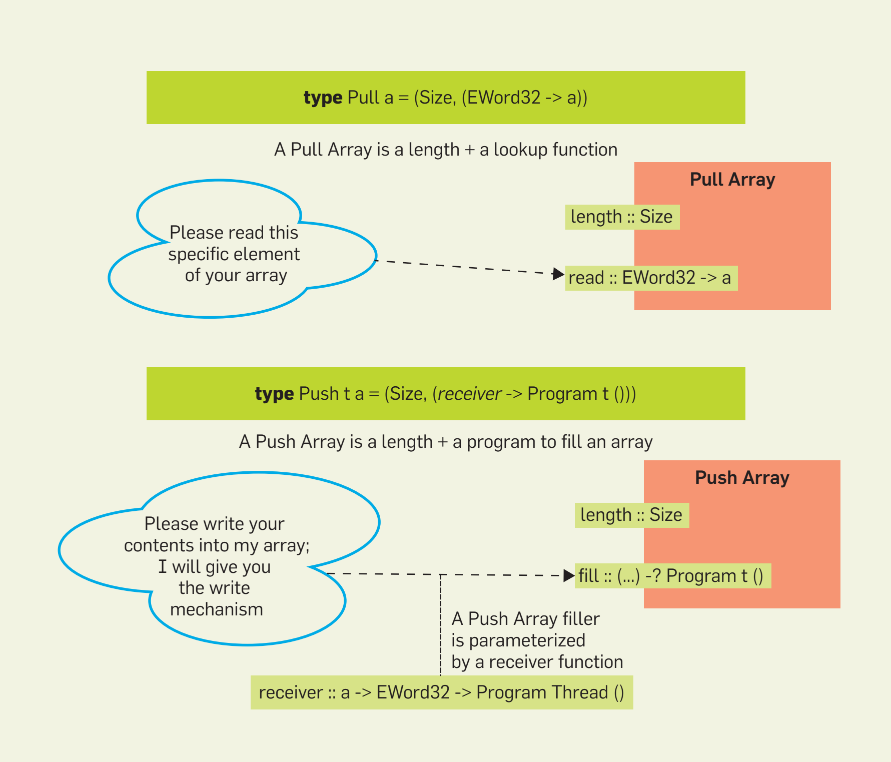
```
vecAdd v1 v2 = zipWith (+) v1 v2
```

The `vecAdd` function takes two arrays, `v1` and `v2`, as input and performs elementwise addition using the `zipWith` library function (`zipWith` is just a `map` over two arrays; see Figure 1 for a selection of library functions). `vecAdd` is not a complete Obsidian program, however; information about how to map this program onto the GPU hierarchy is needed and will be provided in the program's types.

Figure 1. A selection of operations on pull and push arrays.

Operation	Input	Output	Description
map function	pull array	pull array	apply function to each element of input array
map function	push array	push array	apply function to each element of input array
zipWith function	two pull arrays	pull array	pairwise version of map
reverse	pull array	pull array	reverses a pull array
splitUp n	pull array	nested pull arrays	splits a pull array into chunks of size n
halve	pull array	pair of pull arrays	splits a pull array in the middle
push	pull array	push array	converts from pull to push representation
force	push array	pull array	makes array manifest in memory
forcePull	pull array	pull array	makes array manifest in memory

Figure 2. The interface between push/pull arrays and their clients.


type Pull a = (Size, (EWord32 -> a))
A Pull Array is a length + a lookup function
Pull Array
length :: Size
Please read this specific element of your array
read :: EWord32 -> a
type Push t a = (Size, (receiver -> Program t ()))
A Push Array is a length + a program to fill an array
Push Array
length :: Size
Please write your contents into my array; I will give you the write mechanism
fill :: (...) -? Program t ()
A Push Array filler is parameterized by a receiver function
receiver :: a -> EWord32 -> Program Thread ()

To give precise types to operations such as `vecAdd` requires an understanding of deferred arrays described in the first section. Specifically, there are *push* and *pull* array variants. The `vecAdd` program operates on pull arrays. Its type is:

```
vecAdd :: Pull EFloat
       -> Pull EFloat
       -> Pull EFloat
```

That is, `vecAdd` takes two pull arrays of `EFloat` values as input and returns a pull array. (Values with types prefixed by `E` are actually expression trees, as we will explain.) Pull arrays are implemented as a length plus a function from index to value:

```
type Pull a = (Size, (EWord32 -> a))
```

This representation of arrays provides fusion of operations for free. For example, with the above definition of a pull array, the following equations show how a function mapped over a pull array is actually absorbed (composed) into the function-based representation it already uses:

```
map f arr == map f (sz,g) == (sz, f ∘ g)
```

No intermediate data is written to memory for the output of `g`, but in some cases it is desirable to write the elements to memory—for example, revisiting the example from the first section, but now assuming that `f2` is expensive in the following code:

```
let arr2 = map f2 arr1
    sum = reduce (+) arr2
    prd = reduce (*) arr2
...
```

Here `arr2` is used twice, and the programmer should have the power to ensure that `arr2` is computed and fully stored in memory. This is where `force` functions come in:

```
do arr2 <- forcePull (map f2 arr1)
   let sum = reduce (+) arr2
       prd = reduce (*) arr2
   ...
```

The reason for switching to Haskell's do notation is that forcePull is a *monadic* function. Monads are Haskell's way of encoding side effects, and in this program the monad in question is Obsidian's Program monad, which roughly corresponds to CUDA code. The return type of forcePull is Program t (Pull a), where t is a parameter designating a level in the GPU hierarchy (Thread, Warp, Block, or Grid).

With the Program nomad, we can define push arrays as well:

```
type Push t a =
 (Size,
  Receiver (a -> Program t ())
type Receiver a =
 (a -> EWord32 ->
  Program Thread ())
```

Like pull arrays, push arrays consist of a length and a function. The difference is that the function in the push array (the filler function) results in a program that encodes an iteration schema at level t in the hierarchy. The argument to the filler function is itself a function (the receiver function). A pull array fulfills requests for specific elements, whereas a push array only allows a bulk request to push *all* elements via a receiver function. This relationship is pictured in Figure 2. When invoked, the filler function creates the loop structure, but it inlines the code for the receiver inside that loop. A push array with its elements computed by f and receiver rcv might generate: for(i∈[1,*N*]) { rcv(i,f(i)); }

More specifically, when forcing a push array to memory, each invocation of rcv would write one memory location, A[i] = f(i).

The reason Obsidian has both pull and push arrays is related to performance. Some operations are very easy to express on pull arrays and result in good code, such as map, zipWith and permutations. An operation such as append, however, which takes two pull arrays and concatenates them, leads to a conditional within the lookup function. When forcing such a concatenated array, this conditional will be executed in every iteration of the generated for loop (or by each thread). Push arrays, on the other hand, encode their own iteration schema, so rather than executing a conditional, when concatenating push arrays their inherent loop schemas are executed in sequence, neither loop executing a conditional in each iteration.

Push and Pull Conversion

The result of force (for push arrays) and forcePull (for pull arrays) is always a pull array. That means a push array can be converted to a pull array by application of force. Applying force always has a cost in memory, however, and it realizes the full cost of computing all elements.

Going in the other direction—converting from pull array to push array—requires a function called push. This function does not have any cost directly associated with it. It merely changes the representation (though switching to the push representation means losing the ability to compute only *part* of the array). Note, however, that pull arrays do not have the t parameter like push arrays—they are hierarchy-level agnostic. Using push on a pull array and fixing the t parameter of the result locks the array in a given level of the hierarchy, as in the following:

```
vecAdd :: Pull EFloat
       -> Pull EFloat
       -> Push Grid EFloat
vecAdd v1 v2 =
 push (zipWith (+) v1 v2)
```

Now the vecAdd program is complete: push converts the result array to a push array, and the result type is fixed as Push Grid, so the iteration becomes Grid-level parallel.

Design-space exploration. Even a simple operation such as reduction requires design-space exploration for good performance on the GPU. Sequential reductions per thread can be combined with parallel binary-tree-shaped reduction over multiple threads in a block. Both the degree of sequential work and the number of threads used for the parallel work can be varied. This section describes a local (on-chip, shared-memory) reduce algorithm. It performs one reduction per GPU block. This can serve as a building block in a full-scale reduction algorithm over all the GPU threads.

The code shown in Figure 3 implements a recursive parallel reduction. In each step, the input array is split in the middle, and the elements from the two resulting arrays are added together pairwise, resulting in an intermediate array half as long. The intermediate array is forced to memory using forcePull, writing to shared memory using one thread per element (in this case, at the Block level).

Next, many instances of the block-level reduction algorithm are combined to form a grid computation. The code noted in Figure 4 distributes the computation over multiple blocks using map and pConcat. The pConcat function is for *nested parallelism*; it concatenates the results computed in

Figure 3. A block-local reduction function.

```
reduceLocal :: Scalars a => (a -> a -> a) -> Pull a -> Push Block a
reduceLocal f arr = singletonPush (loop arr)
  where loop arr | len arr == 1 = return (arr ! 0)
                 | otherwise =
                    do let (a1,a2) = halve arr
                       arr' <- forcePull (zipWith f a1 a2)
                       loop arr'
```

Figure 4. Run many block-local reductions in parallel over chunks of 4096 elements.

```
reduceGrid :: Scalars a => (a -> a -> a) -> Pull a -> Push Grid a
reduceGrid f arr = pConcat (map (reduceLocal f) chunks)
  where chunks = splitUp 4096 arr
```

each block and is in charge of the parallel distribution of work across blocks. (A related function is `sConcat`, which performs sequential work within one block or one thread.)

The `forcePull` function prevents fusion of operations, so leaving it out in some stages of the reduction is a way of trading off sequential and parallel execution. An `optForce` *n* function could be written that forces only arrays shorter than *n* elements:

```
optForce n arr =
  if len arr <= n
  then forcePull arr
  else return arr
```

The `optForce` function could replace `forcePull` in `reduceLocal`. The `n` parameter in `optForce` is one possible tuning parameter to add to `reduceLocal`; another is the `splitUp` factor (`4096` above). By parameterizing on the force cutoff and the `splitUp` factor, a family of different reduction codes can be generated from the same EDSL description.

Obsidian provides yet another tuning parameter by default: the number of real GPU threads to allow per block. This parameter is given by the Obsidian user to the code generator and can range over the number of threads per block supported by CUDA (1–1,024). Given the execution model of GPUs, however, only multiples of the warp size (32) make sense.

Allowing the parameters to vary over the following ranges:

- `splitUp` factor: $s \in \{32,64,\ldots,8192\}$
- `optForce` cut-off: $c \in \{32,64,\ldots,s\}$
- Real Threads: $t \in \{32,64,96,\ldots, min(1024,s/2)\}$

would create 929 variants of the reduction code for auto-tuning. For benchmark results for reduction kernels, refer to the authors' recent technical report.[20]

Generating CUDA Code from the EDSL

Up to this point we have seen the programmer's view of Obsidian, with dips into implementation details here and there. But how does it all work? The first article in this series (Gill) shows much of the needed infrastructure: overloading, reification, and expression data types. Obsidian is no different; the `EWord32` and `EFloat` types seen throughout this article correspond to expression trees. The `Program` monad is reified into abstract syntax representing a program in an imperative, CUDA-like language.

Even a simple operation such as reduction requires design-space exploration for good performance on the GPU.

One new concept presented here, compared with Gill's article, is the use of pull and push arrays. These deferred arrays rely on extracting program fragments as abstract syntax (that is, deep embedding), but the functions that represent push and pull arrays are not themselves captured in abstract syntax. Rather, they are like macros that desugar during Haskell execution, *before* the Obsidian CUDA-emitting compiler is invoked (shallow embedding).[17]

An outline of CUDA code generation would include:

- *Function reification.* EDSL functions (such as `vecAdd`) are applied to a symbolic array, causing it to be evaluated as a Haskell program and yielding its result. The functions Obsidian can reify have push arrays as result. After the application to a symbolic array, therefore, you have a push array. The push array is a function that generates a `Program`, and applying that push array's filler function to yet another symbolic input extracts a final, complete `Program`.
- *Monadic reification.* Because the `Program` type is a monad, a monad reification technique is applied (see Gill), and the result is an AST (abstract syntax tree) describing a program in a CUDA-like imperative language. This AST, however, has explicit parallel for loops rather than CUDA's implicit ones.
- *Shared memory analysis.* All intermediate arrays (resulting from `force`) have unique names in the CUDA-like AST, but they still must be laid out in shared memory. Liveness analysis finds the first and last use of each array, and a memory map is generated. Named arrays in the AST are replaced by exact locations in shared memory.
- *CUDA generation and thread virtualization.* This is the final step in code generation. Most aspects of the CUDA-like AST translate directly into CUDA source. The explicit parallel loops, which may have larger iteration spaces than the number of actual CUDA threads, need some further conversion. If the AST con-

tains a loop, “`parFor(i∈[1,512]){body})`,” but the code generator was instructed to use only 256 threads, then this `parFor` will happen in two stages, implemented by injecting an additional sequential loop.

Conclusion

This article has highlighted a number of potential benefits of DSL technology. If you are going to give embedded DSLs a try, however, there are some downsides to watch out for as well. First, error messages can be problematic, as these are expressed in the terminology of the host language. If Obsidian programmers try to use some feature at the incorrect level in the GPU hierarchy (for example, a too-deep nesting of parallel `for` loops by applying too many `pConcats`), then the error message will likely state there are missing instances of some Haskell class, rather than explaining why what they tried cannot be done on a GPU. Furthermore, new DSLs naturally lack the library and tooling ecosystems of large established languages.

Using an EDSL such as Obsidian, which exposes underlying architectural structure, allows control over the details that determine performance. Choosing exactly what to abstract and what to expose, however, remains important. For example, in Obsidian programmers can trade off sequential and parallel work and use shared memory (via `force`), but the details of shared-memory layout of arrays, or managing their lifetimes in that memory, are automated. Further, just as important is what the DSL *prevents* the user from doing. In Obsidian, programmers are gently, but firmly, prevented from writing DSL code that would produce CUDA that does not match what the GPU can cope with, such as the too-deep nestings just mentioned.

Writing CUDA code and hand-tuning for performance is often a tedious task. Design decisions made early may be difficult to revert and require much of the code already written to be replaced. It is thus appealing to generate variants of CUDA code from a highly parameterized description in a DSL, and then select the best performing, empirically. Our experiments in previous work[20] show the best variant of particular code may differ from one GPU to another.

How much work does it take to produce a code-generating DSL such as Obsidian? In Obsidian’s case there have been years of experiments and redesigns—leading to a Ph.D.![18] But do not let that put you off. The current version is relatively small (2,857 lines in the language implementation and 1,629 lines in the compiler part), and it is freely available to study and copy.[19] Further, modern libraries for parsing, code generation from templates (for example, http://hackage.haskell.org/package/language-c-quote), and compiler construction[9] make constructing DSLs easier than expected.

So, the next time you find yourself with repetitive low-level code, why not consider an embedded code-generating DSL instead?

Acknowledgments

Push arrays were invented by Koen Claessen. We thank Andy Gill for his valuable feedback on this article. Work on Feldspar and Obsidian was supported by the Swedish Foundation for Strategic Research in the Resource Aware Functional Programming (RAW FP) project and by the National Science Foundation award 1337242.

Related articles on queue.acm.org

Purpose-Built Languages
Mike Shapiro
http://queue.acm.org/detail.cfm?id=1508217

The Ideal HPC Programming Language
Eugene Loh
http://queue.acm.org/detail.cfm?id=1820518

Creating Languages in Racket
Matthew Flatt
http://queue.acm.org/detail.cfm?id=2068896

References

1. Axelsson, E., Claessen, K., Sheeran, M., Svenningsson, J., Engdal, D. and Persson, A. The design and implementation of Feldspar, an embedded language for digital signal processing. *Implementation and Application of Functional Languages.* Springer Verlag, 2011.
2. Catanzaro, B., Garland, M., Keutzer, K. Copperhead: Compiling an embedded data parallel language. In *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, (2011), 47–56.
3. Chafi, H., Sujeeth, A.K., Brown, K.J., Lee, H.J., Atreya, A.R. and Olukotun, K. A domain-specific approach to heterogeneous parallelism. *ACM SIGPLAN Notices 46* (2011), 35–46.
4. Chakravarty, M.M.T., Keller, G., Lee, S., McDonell, T.L. and Grover, V. Accelerating Haskell array codes with multicore GPUs. In *Proceedings of the 6th Workshop on Declarative Aspects of Multicore Programming*, (2011).
5. Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R. R., Bradshaw, R., Weizenbaum, N. 2010. FlumeJava: easy, efficient data-parallel pipelines. In *ACM SIGPLAN Notices* 45: 363–375.
6. Chhugani, J., Nguyen, A.D., Lee, V.W., Macy, W., Hagog, M., Chen, Y.-K. and Baransi, A., Kumar, S. and Dubey, P. Efficient implementation of sorting on multicore SIMD CPU architecture. In *Proceedings of the VLDB Endowment 1*, 2 (2008), 1313–1324.
7. Claessen, K., Sheeran, M., Svensson, B.J. Expressive array constructs in an embedded GPU kernel programming language. In *Proceedings of the 7th Workshop on Declarative Aspects and Applications of Multicore Programming*, (2012).
8. Filipovič, J., Madzin, M., Fousek, J. and Matyska, L. Optimizing CUDA code by kernel fusion—application on BLAS. (2013); arXiv preprint arXiv: 1305.1183.
9. Keep, A.W., Dybvig, R.K. A nanopass framework for commercial compiler development. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, (2013), 343-350.
10. Keller, G., Chakravarty, M.M.T., Leshchinskiy, R., Peyton Jones, S. and Lippmeier, B. Regular, shape-polymorphic, parallel arrays in Haskell. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, (2010).
11. Newburn, C.J. et al. Intel’s array building blocks: a retargetable, dynamic compiler and embedded language. *International Symposium on Code Generation and Optimization*, (2011).
12. NVIDIA. CUDA C Programming Guide, (2013); http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.
13. NVIDIA. Nvidia’s Next Generation CUDA Compute Architecture: Kepler GK110, (2012); http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.
14. Püschel, M. et al. Spiral: Code generation for DSP transforms. In *Proceedings of the IEEE 93*, 2 (2005), 232-275.
15. Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F. and Amarasinghe, S. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image-processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, (2013), 519-530.
16. Scholz, S.-B. 2003. Single assignment C: Efficient support for high-level array operations in a functional setting. *Journal of Functional Programming 13*, 6 (2003), 1005-1059.
17. Svenningsson, J. and Axelsson, E. Combining deep and shallow embedding for EDSL. In *Trends in Functional Programming*, H.-W. Loidl and R. Pea, eds. *Lecture Notes in Computer Science 7829* (2013), 21-36. Springer, Berlin/Heidelberg.
18. Svensson, B.J. Embedded languages for data-parallel programming. Ph.D. thesis, 2013. Department of Computer Science and Engineering, Chalmers University of Technology.
19. Svensson, B.J. Obsidian GitHub Repository, 2014; https://github.com/svenssonjoel/Obsidian.
20. Svensson, B.J., Sheeran, M., Newton, R.R. A language for nested data parallel design-space exploration on GPUs. Technical Report 712 (2014). Indiana University; http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR712.
21. Tang, Y., Chowdhury, R.A., Kuszmaul, B.C., Luk, C.-K. and Leiserson, C.E. The pochoir stencil compiler. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*, (2011), 117-128.
22. Thies, W., Karczmarek, M. and Amarasinghe, S. StreamIt: A language for streaming applications. In *Compiler Construction*, (2002), 179-196. Springer.

Bo Joel Svensson is currently a postdoc at Indiana University working with Ryan R. Newton on a project related to EDSLs and parallelism.

Mary Sheeran is a professor in the software technology division and member of the functional programming group at Chalmers University of Technology, Gothenburg, Sweden.

Ryan R. Newton joined Indiana University in 2011, where his research focuses on language-based approaches to the programming challenges posed by future architectures.

DOI:10.1145/2602575

EPIC helps assess cyberthreats against the cyber and physical dimensions of networked critical infrastructures.

BY CHRISTOS SIATERLIS AND BÉLA GENGE

Cyber-Physical Testbeds

MODERN SOCIETIES DEPEND on the quality and reliability of the services provided by networked critical infrastructures (NCIs). Physical infrastructures, including transportation systems, electricity grids, and telecommunication networks, deliver fundamental services for the smooth functioning of the economy and for the lives of all citizens. Accidental or intentional failure represents one of the most important risks faced today.

The past few years have seen a dramatic increase in the use of the information and communication technologies (ICTs) within NCIs. The motivation for companies was mainly to reduce the cost of industrial installations and implement new services (such as remote monitoring and maintenance of infrastructures, energy markets, and the emerging smart grid). Despite clear advantages, the downside is widespread use of standard technology components exposes NCIs to significant but common cyberthreats; for instance, deliberate attacks through computer malware[6] or unintentional threats due to misconfiguration and software bugs[5] can lead to severe service outages. This was also highlighted by several studies and reports concerning security of supervisory control and data acquisition, or SCADA, systems,[6,15] which represent core NCI infrastructure, monitoring and controlling physical processes. They consist mainly of actuators, sensors, and hardware devices that perform physical actions (such as open a valve), as well as the ICT devices and software that monitor and control physical processes. Unlike traditional ICT systems, where the effects of disruptive cyberattacks are generally limited to cyber operations, in the context of critical infrastructure assets, such attacks can result in the loss of vital services (such as transportation and water and gas supply). Assessing the effect of cyberthreats against both the physical and the cyber dimensions of NCIs requires an accurate scientific instrument for conducting experimental tests and taking measurements.

Cyber-physical testbeds that actively support the scientific method are an

» key insights

- EPIC supports execution of scientifically rigorous cyber-physical security experiments, ensuring fidelity, repeatability, measurement accuracy, and safe execution.
- EPIC recreates interdependencies between critical infrastructure components by leveraging real networks, protocols, and software involved in the cyber and software simulators for the physical dimensions of the infrastructures.
- EPIC represents a new class of scientific instrument—cyber-physical testbeds—suitable for testing the effects of cyberattacks, validating novel countermeasures, and training users in scenarios involving a range of infrastructures, from local physical processes (such as power plants) to national grids and transportation systems.

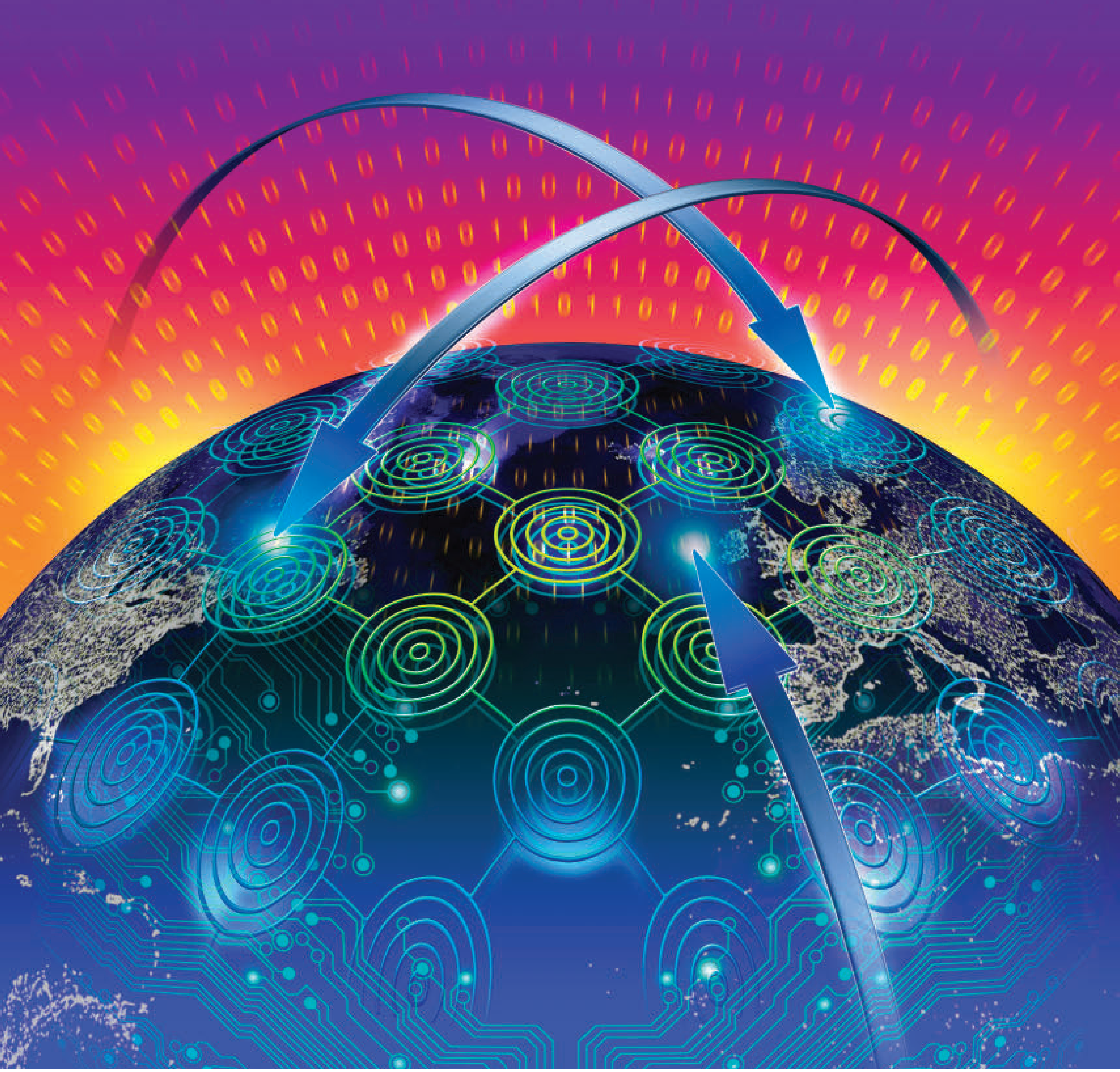


ILLUSTRATION BY ROY WIEMANN

example of such an instrument. Testbed development may leverage real systems, emulators, or software simulators. Unfortunately, experimentating with production systems in security and resilience tests involves the risk of side effects to mission-critical services.[10] Likewise, dedicated experimentation infrastructure with real components involves safety risks[10] and high installation costs.[11] Software-based simulation is an efficient way to study physical systems, offering low-cost, fast, accurate analysis. However, it has limited applicability in the context of cybersecurity in light of the diversity and complexity of computer networks. Software simulators effectively model normal network conditions but fail to capture the way computer networks fail.[7] On the other hand, in many cases, emulators capture not only whether a system will fail but also how it will fail.

To address the need for cyber-physical testbeds, we propose a scientific instrument called the Experimentation Platform for Internet Contingencies, or EPIC, we developed to provide accurate, repeatable assessments of the effect cyberattacks might have on the cyber and physical dimensions of NCIs. To model the complexity of NCIs, EPIC uses a computer testbed based on Emulab (an advanced software suite for creating emulation testbeds)[20,24] to recreate the cyber elements of an NCI and SSims for the physical components. (The term Emulab refers to both a facility at the University of Utah, Salt Lake City, and to a type of software.)

Motivation

A major limitation of existing testbeds is the inability to run security experiments on multiple heterogeneous

Table 1. Testbed features and cost-effectiveness compared: ••• = strong support; •• = moderate support; and • = weak support for a specific feature.

	Fidelity		Repeatability		Measurement Accuracy		Safety			
	Cyber	Physical	Cyber	Physical	Cyber	Physical	Cyber	Physical	Cost Effectiveness	Multiple Critical Infrastructures
Real cyber, real physical										
NSTB[23]	•••	•••	••	•	••	••	•••	•	•	•
ENEL-JRC[15]	•••	•••	••	•	••	••	•••	•	•	•
Real cyber, sim physical										
EPIC	•••	••	•••	•••	•••	•••	•••	•••	••	•••
DEFT[25] (DETER+VPST)	•••	••	•••	•••	•••	•••	•••	•••	•	•
PowerCyber[11]	•••	••	••	•••	••	•••	•••	•••	••	•
Real and sim cyber, real physical										
TUB[8]	••	•••	••	•	••	••	•••	•	•	•
Real and sim cyber, sim physical										
VCSE[14]	••	••	••	•••	••	•••	•••	•••	••	•••
Sim cyber, sim physical										
SCADASim[17]	•	••	•••	•••	•••	•••	•••	•••	•••	•••
HLA[16]	•	••	•••	•••	•••	•••	•••	•••	•••	•••

Table 2. Typical experiments performed through EPIC.

Objective	Nodes	Attack Type	Physical Process	Outcome
Evaluate effect of network parameters—delay, packet loss, and segmentation—on cyber attacks targeting NCIs	6–15	Spoofing	Power and chemical plants	Experiments proved communications delays and packet losses have an insignificant effect on cyber attacks, though physical-process-aware network segmentation increases NCI resilience. We thus developed and validated a novel segmentation approach.
Validate novel approach for anomaly detection in cyber-physical systems	21	DDoS and spoofing	Power grid	The experiments led to validation of a proof-of-concept prototype showing superior performance of combined cyber-physical anomaly detection, separating cyber from physical over traditional approaches.
Evaluate effect of operational decisions and their cascading effects on interdependent NCIs	12	Coordinated attack	Power grid and railroad transportation	They showed propagation of cyber disturbances (such as loss of monitoring and control) from one NCI to other dependent NCIs and that NCI operators must collaborate to ensure global stability of interconnected NCIs.
Analyze effect of operator reactions and coordination in well-known (YouTube) BGP route hijacking incidents	32	Hijacking and man-in-the-middle		They highlighted the importance of tools and mechanisms for fast discovery of BGP hijacking events, well-trained operators with security expertise, and a trusted medium to support communication and coordination among network service providers.

NCIs; NCIs are highly interconnected and interdependent, meaning a single failure within one NCI could have a cascading effect on others; for example, the collapse of India's northern electricity grid in July 2012 affected more than 600 million people and led to loss of power for transportation, health care, and many other services. Such scenarios (which can also be caused by cyberattacks) must be recreated, analyzed, and understood in a laboratory environment in order to develop the necessary security measures that can be applied in real-world settings.

By recreating key connections between the cyber and the physical in NCIs, EPIC provides a diverse palette of research applications. Along with vulnerability testing, effect analysis, and validation of different techniques, EPIC also provides tools for closing an important loop in cyber-physical experimentation—the human operator. In the NCI context, human operators help ensure the stability and normal functioning of physical processes. Human operators can interact directly with EPIC as part of an experiment or be simulated by modeling their standard operating procedures. Either way, EPIC can be used to build complex experiments for testing the effect of commands issued by human operators on physical processes or measure the reaction of human operators to the changes in the state of some physical process. EPIC thus brings an important development in experimentation testbeds through accurate experiments that are closer to real NCI operation.

Testbed requirements. A cyber-physical testbed must be compatible with and actively support the scientific method, enabling researchers to apply rigorous scientific methods by ensuring the fidelity, repeatability, measurement accuracy, and safe execution of experiments:[20]

Fidelity. Experimentation testbeds must be able to reproduce as accurately as possible the real system under study. However, in many cases reproducing all details of a real system in an absolute way might not be necessary or even possible. It is thus preferable for an experimental platform to offer an adjustable level of realism, meaning researchers can use the level of detail that is sufficient to test the experimental hypothesis; for example, one experiment might need to reproduce a network at the physical layer using real routers, while for another a software router might be enough. The concept of an adjustable level of realism means having the option of using real hardware when really needed or emulators, simulators, or other abstractions when not needed.

Repeatability. This requirement reflects the need to repeat an experiment and obtain the same or statistically consistent results. Repeatable experiments require a controlled environment, but to achieve them researchers must first define the experiment's initial and final state, as well as all events in between. To reproduce a previously stored experiment scenario researchers must be able to set up the experimental platform in the initial state and trigger all necessary events in the right order and time of occurrence.

Measurement accuracy. Experiments should be monitored but not interfere with the experiment in a way that might alter its outcome. Needed therefore are separation of control, measurement, and experiment processes.

Safe execution. In most cases security experiments assume the presence of adversaries employing malicious software to achieve their goals. The effect of the software is often unpredictable, including disruptive effects on physical systems. Experiments must recreate such instances without jeopardizing the physical testbed itself or the researchers.

Before choosing the simulation step, EPIC researchers must verify the output of real-time simulation reproduces as accurately as possible its counterpart real-world process.

Existing approaches. To assess the state of the art, we performed a literature review and evaluated the features of available testbeds against the previously defined set of requirements (see Table 1).

The U.S. National SCADA TestBed (NSTB) program run by the U.S. Department of Energy[23] constitutes a national collaborative laboratory project intended to support industry and government efforts to enhance the cybersecurity of industrial installations, providing a range of facilities to recreate real-world systems, from generation to transmission, including real powergrid components, as well as industry-specific software products.

Although the NSTB helped identify vulnerabilities and harden control-system protection mechanisms, the cost of deploying a similar installation limits its practical application in multi-domain heterogeneous cyber-physical systems.

A collaborative effort between Enel S.p.A., the largest power company in Italy, and the Joint Research Centre, Italy, the scientific and technical arm of the European Union's executive body, led to development in 2009 of a protected environment recreating the physical characteristics of a real turbogas power plant.[15] The testbed accurately reproduces both cyber and physical characteristics of a typical power plant, including a scaled-down physical process, typical field networks, process network, security zones, horizontal services, corporate domain, and standard software. The testbed is used to analyze attack scenarios and test countermeasures in a safe environment. Unfortunately, the high fidelity of a pure physical testing environment is offset by poor flexibility and the high cost of maintenance of similar architectures.

The Emulab-based cyber-DEfence Technology Experimental Research (DETER) testbed[2] provides repeatable security-related experiments, part of the DETER Enabled Federated Testbeds (DEFT) for interconnecting geographically distributed testbeds in cyber-physical experimentation. Within the DEFT consortium DETER was interconnected[25] in 2009 with the Virtual Power System Testbed (VPST) developed by the University of Illinois.[3] VPST provides simulation capabilities for electricity grids through real-time simulators (such as PowerWorld, a proprietary power-system simulator), extending DETER capabilities to experimentation with cyber-physical systems.

The key difference between EPIC and DEFT is EPIC provides a scalable cost-effective solution for experimenting with multi-domain heterogeneous physical processes (through its software simulators), while DEFT is more focused on a specific infrastructure (such as the power grid). EPIC can also be viewed as complementary to the DEFT initiative since the software simulators developed for EPIC are easily reused through DETER.

The PowerCyber testbed developed at Iowa State University[11] in 2013 integrates SCADA-specific hardware and software with real-time digital simulators to simulate electrical grids. It uses virtualization techniques to address scalability and cost and the Internet-Scale Event and Attack Generation Environment project also developed at Iowa State University for wide-area network emulation. The testbed further provides non-real-time simulation capabilities, primarily for simulating larger systems and for performing state estimation and contingency analysis.

An approach using real components for the physical dimension and partly simulated components for the cyber dimension comes from the Tsinghua University of Beijing,[8] using real SCADA control servers and the NS-2 network simulator combined with real control hardware and field devices. This testbed was designed to determine the effect of cyberattacks on the SCADA system, including packet forging, compromised access-control mechanisms, and compromised SCADA servers. Although it provides reliable experimental data, since almost everything in it is real, it is unable to support tests on large infrastructures (such as a national electric grid).

Sandia National Laboratory developed the Virtual Control System Environment (VCSE) testbed[14] to explore vulnerabilities, train operators, and validate mitigation techniques. VCSE employs computer-network performance-analysis software called OPNET to integrate real devices with simulated networks and PowerWorld as its power system simulator. VCSE also incorporates Umbra, Sandia's patented framework that provides a centralized environment for monitoring and controlling multiple simulated components.

NCIs are highly interconnected and interdependent, meaning a single failure within one NCI could have a cascading effect on others.

The SCADASim framework[17] developed at the Royal Melbourne Institute of Technology, Melbourne, Australia, provides predefined modules for building SCADA simulations, employing the OMNET++ discrete event simulation engine to recreate typical SCADA components while providing an underlying inter-model communications layer. SCADASim supports integration with real devices through modules implementing industry-standard protocols. It can be used to develop a range of SCADA simulations and evaluate the effect of cyberattack scenarios on communications, as well as on the normal functioning of physical processes.

Finally, the system-of-systems approach to testbed development at the Swiss Federal Institute of Technology, Zurich,[16] uses the High Level Architecture simulation standard to provide a multi-domain experimentation environment for interconnecting simulators from multiple domains. The testbed supports exploration of what-if scenarios in the context of complex interdependencies between critical infrastructures. Unfortunately, such an approach might be effective on interdependency studies but be unable to recreate the cyber layer accurately.

EPIC Overview

EPIC architecture involves an emulation testbed based on Emulab software[20,24] to recreate the cyber dimensions of NCIs and software simulation for the physical dimension. By employing an emulation-based testbed, EPIC ensures fidelity, repeatability, measurement accuracy, and safety of the cyber layer, an approach well established in the field of cyber security[2] chosen to overcome major difficulties trying to simulate how ICT components behave under attacks or during failures. EPIC uses simulation for the physical layer, since it provides an efficient, safe, low-cost approach with fast, accurate analysis capabilities. Although it weakens the fidelity requirement, software simulation enables disruptive experiments on multiple heterogeneous physical processes. Moreover, we find complex models of several physical systems in the literature, and the behavior of real physical systems can be reproduced accurately by integrating them into software simulators; one example is the energy sector, where simulation is so accurate and trusted it is commonly used to aid decision making by operators of transmission systems.

Recreating cyber systems. Emulab[24] is an advanced software suite for emulation testbeds, with many private installations worldwide and support from multiple universities. In 2009, we developed our first installation using the Emulab architecture and software, now under continuous development and expansion through the Emulab architecture and software (see Figure 1a). Adopting Emulab in EPIC, we automatically and dynamically map physical components (such as servers and switches) to a virtual topology; that is, Emulab software configures physical topology to emulate the virtual topology as transparently as possible. The basic Emulab architecture consists of two control servers: a pool of physical resources used as experimental nodes (such as generic PCs and routers) and a set of switches interconnecting the nodes. Emulab software provides a Web interface to describe the steps that define the experiment life cycle within the EPIC testbed:

Virtual network topology. EPIC researchers must first create a detailed description of the virtual network topology, the "experiment script"; using a formal language for experiment setup eases recreation of a similar arrangement by other researchers who might want to reproduce our results;

Emulab software. Experiments are instantiated through Emulab software, which automatically reserves and allocates the physical resources needed from the pool of available components;

Experimental nodes. The software configures network switches to recreate virtual topology by connecting experimental nodes through multiple virtual local-area networks, then configures packet capturing of predefined links for monitoring purposes; and

Defined events. Experiment-specific software (such as simulators) is launched automatically through events defined in the experiment script or manually by logging in to each node.

Recreating physical systems. The software units that recreate physical systems within EPIC are outlined in Figure 1b. Physical process models are built in Matlab Simulink, from which the corresponding C code is generated through Simulink Coder. The generated code is then integrated into the software simulation unit (SSim) to en-

Figure 1. EPIC testbed architecture: (a) overview and experimentation steps; and (b) software modules, including SSims and proxy units.

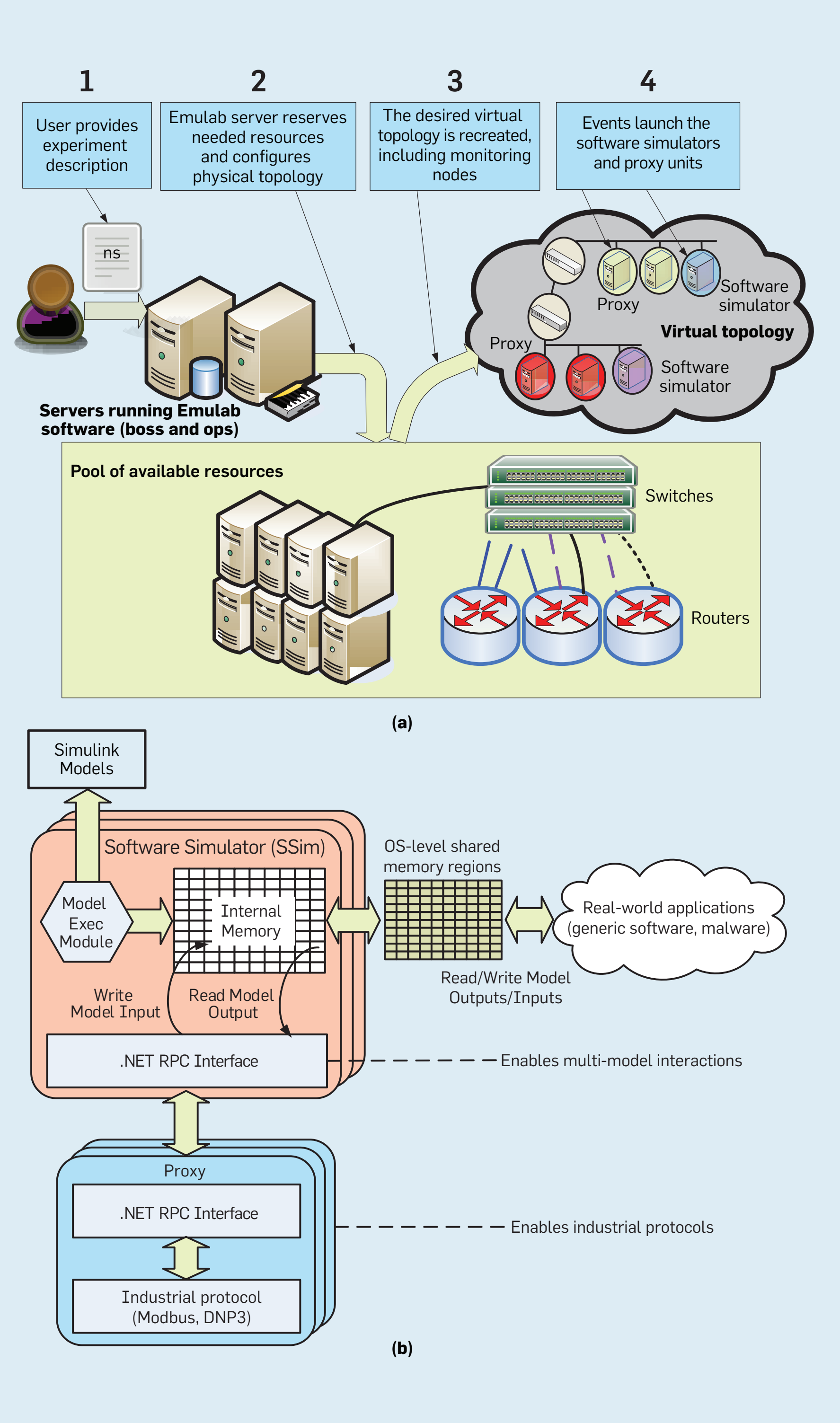
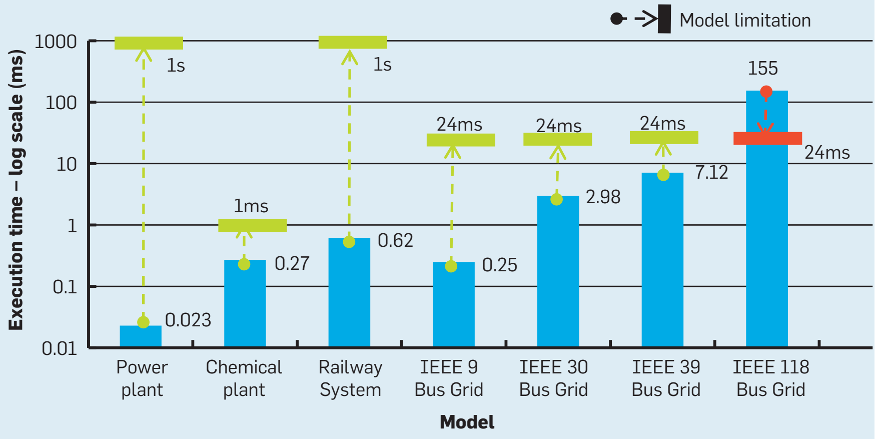

1
User provides experiment description
2
Emulab server reserves needed resources and configures physical topology
3
The desired virtual topology is recreated, including monitoring nodes
4
Events launch the software simulators and proxy units
ns
Proxy
Software simulator
Virtual topology
Proxy
Software simulator
Servers running Emulab software (boss and ops)
Pool of available resources
Switches
Routers
(a)
Simulink Models
Software Simulator (SSim)
Model Exec Module
Internal Memory
Write Model Input
Read Model Output
.NET RPC Interface
OS-level shared memory regions
Real-world applications (generic software, malware)
Read/Write Model Outputs/Inputs
Enables multi-model interactions
Proxy
.NET RPC Interface
Industrial protocol (Modbus, DNP3)
Enables industrial protocols
(b)

Figure 2. Execution time on a 2.8GHz CPU and limitations of various models; EPIC enables experimentation with power plants, chemical plants, railway systems, and power grid models from a suite of standard IEEE models.

The red line on the IEEE 118 bus model highlights real-time simulation requirements are not met since model execution time exceeds the maximum simulation step, or the model-specific time interval or granularity in which the model must run at least once, as allowed by model dynamics.



able real-time interaction of simulated processes with the rest of the emulation testbed. At its core, the SSim unit binds the cyber and the physical layers. Viewed from the SSim's perspective, models are the equivalent of black boxes, with inputs and outputs dynamically mapped to an internal memory region. Values written into this region are copied to the model's inputs, while model outputs are copied back to internal memory. This way, EPIC enables experimentation with a range of physical processes without having to provide details of their content. To enable interdependency studies on multiple NCIs, SSim implements a remote procedure call (RPC) interface accessible to other SSim instances. RPCs provide access to the internal memory region and consequently to the model's inputs and outputs, enabling real-time interaction between models. Moreover, EPIC supports industrial protocols (such as Modbus) through Proxy units that translate calls between SSim and other units, including servers in industrial installations.

Integrating real-world hardware and software. Since almost all its components are real, EPIC supports any software that usually runs on a regular PC and can integrate practically any hardware equipped with an Ethernet network interface; for instance, the EPIC-based testbed includes real control hardware and real industrial software that enable studies of specific industrial architectures. Interaction between real-world software and EPIC software units is achieved in several ways: First, real software interacts with the simulated models through industrial protocols (such as Modbus). Modbus calls are sent to a Proxy unit that forwards them as RPCs to the SSim unit. Another way to interact is through operating system-level shared memory. Software units can access a shared memory region mapped to the model's inputs/outputs by the SSim unit, as in Figure 1b. This technique enables interaction with software that does not implement RPC or Modbus, providing a simple way to run more complex security studies.

Real-time simulation on multitasking OS. With real-time simulation, models run in a discrete time domain linked closely to the clock of the operating system, meaning the simulated model runs at the same rate as the actual physical system. EPIC uses generic PCs with multitasking operating systems to run real-time software simulation units. Despite major advantages, our choice of Simulink Coder to produce the simulators imposes several constraints on the simulated models, including model-execution rate, or "simulation step." The model's internal dynamics limit the range of possible simulation steps. Before choosing a simulation step, EPIC researchers must verify the output of real-time simulation reproduces as accurately as possible its counterpart real-world process. In parallel, the model execution time on a specific computer is limited by the model's complexity and the host's processing power. In general, if the model's execution time exceeds its simulation step, real-time simulation is not possible.

Testing the limitations of software simulation in EPIC, we have experimented with several physical processes (see Figure 2). Here, we mention small-scale processes (such as Bell and Åström's oil-fired 160MW electric power plant[1] based on the Sydsvenska Kraft AB plant in Malmö, Sweden, and the Tennessee-Eastman chemical process,[9] likewise based on a real process), though Downs and Vogel[9] introduced minor changes to mask the identity of reactants and products. The railway systems used throughout our experiments are based on the train models proposed by Rios and Ramos[18] that account for several dimensions of real transportation systems (such as weight, speed, acceleration, deceleration, and power consumption).

Then there is the IEEE suite of power grid systems[22] used with EPIC. The nine-bus test case is the Western System Coordinating Council's three-machine nine-bus system, and the 30-bus, 39-bus, and 118-bus test cases represent a portion of the system operated by the American Electric Power utility, Columbus, OH, in early 1960. These test cases constitute realistic models long established in the power-systems community, providing a range of power-system configurations.

Data in Figure 2 reflects how real-time software simulation is well suited to small- and mid-scale models. However, software simulation is limited by CPU speed and model size; for instance, the IEEE 118 bus system is a complex model that includes 54 generators with frequency of 50Hz and a maximum simulation step, or maximum model execution rate, of approximately 24ms. Since the model's execution time on a 2.8GHz CPU is 155ms, real-time simulation is not possible in this case.

This is a well-known limitation of real-time software simulation but can be addressed in several ways; for instance, researchers can leverage parallel processing techniques (such as GPU computing) or dedicated hardware simulators that are more powerful and specifically designed for simulations. However, these approaches are still quite expensive and, in a multi-model environment, could render the cost of the cyber-physical testbed prohibitive.

Implementation details. Installation of EPIC at the European Commission's Joint Research Centre in Ispra, Italy, consists of 120 PCs and approximately 100 virtual machines massively interconnected through two stacks of network switches. In addition, carrier-grade routers (such as Cisco 6503) and industrial-control hardware and software (such as ABB AC 800M control hardware, including Modbus interfaces with control server and human-machine interface software from ABB) are available as experimental resources. We have also developed software units (such as SSim and Proxy) in C# and ported and tested them on Unix-based systems with the help of the Mono platform, enabling cross-platform deployment of C# applications.

Scalability and Applicability

EPIC's ability to recreate both the cyber and the physical dimensions of NCIs provides a spectrum of experimentation options addressing critical infrastructures. Here, we offer an overview of typical experiments conducted with EPIC, along with a full experimental scenario:

Typical experiments. EPIC is used concurrently by many researchers for developing, testing, and validating a range of concepts, prototypes, and tools (see Table 2); see also scientific reports and papers on the EPIC website http://ipsc.jrc.ec.europa.eu/?id=693. The first experiment covered in this article weighed the effect of network parameters on cyberattacks targeting NCIs. We looked into network delay, packet loss, background traffic, and network segmentation on a spoofing cyberattack consisting of an adversary able to send legitimate commands to process controllers. We showed that while communications parameters have an insignificant effect on cyberattacks, physical process-aware network segmentation can yield more resilient systems.

As a general rule, if the model's execution time exceeds its simulation step, real-time simulation is not possible.

The second experiment showed studies help validate the effectiveness of newly proposed protection mechanisms. Using EPIC, we recreated a complex setting, including real networks, protocols, hosts, and routers, hence a realistic environment for the validation of a novel anomaly-detection system. The experiment confirmed researchers can use EPIC to recreate a realistic environment to launch distributed denial-of-service (DDoS) attacks together with spoofing attacks on critical infrastructure assets. These attacks contributed to validation of a novel anomaly-detection system able to efficiently detect anomalies in both the cyber and the physical dimensions of NCIs.

The third experiment focused on the human operator, closing a significant loop in cyber-physical experimentation, including a coordinated cyberattack in which the attacker prevents the normal remote operation of several substations, or "reduction of load," by blocking communications. Consequently, several substations exhibited a significant drop of voltage below nominal operating levels. The experiment showed operational decisions can be the difference between a complete breakdown and system survival, and that collaborations between operators can limit the propagation of cyber disturbances.

The fourth experiment recreated the well-known 2008 YouTube border gateway-protocol-route-hijacking incident,[19] analyzing several hypothetical scenarios. We developed an abstraction of Internet backbone networks in Europe and recreated the incident by replaying real traffic traces. The results highlighted the importance of tools and mechanisms for fast discovery of border-gateway-protocol-hijacking events and, most important, well-trained operators able to communicate over a trusted medium.

Together, these four experiments represent only a fraction of the many directions and applications in which EPIC has proved itself a modern scientific instrument. Moreover, its use is not limited to disruptive experiments but can also take on educational and preparedness activities (such as an environment for cybersecurity exercises).

Illustrative experiment. Here, we illustrate EPIC applicability by exploring the consequences ICT disruptions can have on a critical infrastructure (such as a national power grid). We consider the hypothetical scenario of a cyberattack—specifically a DDoS attack—causing significant telecommunication service degradation propagating across critical infrastructures.

Experiment setup. We thus recreated a typical architecture in which the power grid is controlled remotely (see Figure 3a). Site A located on the operator's premises runs a simplified model of an energy-management system (EMS)[21] to ensure voltage stability. The EMS continuously monitors and adjusts the operational parameters of the power grid model running at Site B located remotely (such as in an electrical substation). The EMS sends commands to emulated control hardware or through proxy units that provide access to the power-grid model inputs and outputs. Communications are provided through the Modbus protocol.

The scientific community uses IEEE electrical-grid models extensively in similar studies since they encapsulate (accurately) the basic characteristics of real infrastructures. We adopted the IEEE 39-bus New England system, including 39 substations and 10 generators. The daily load imposed on our system derives from real data,[13] and EMS intervention is required to keep the grid stable.

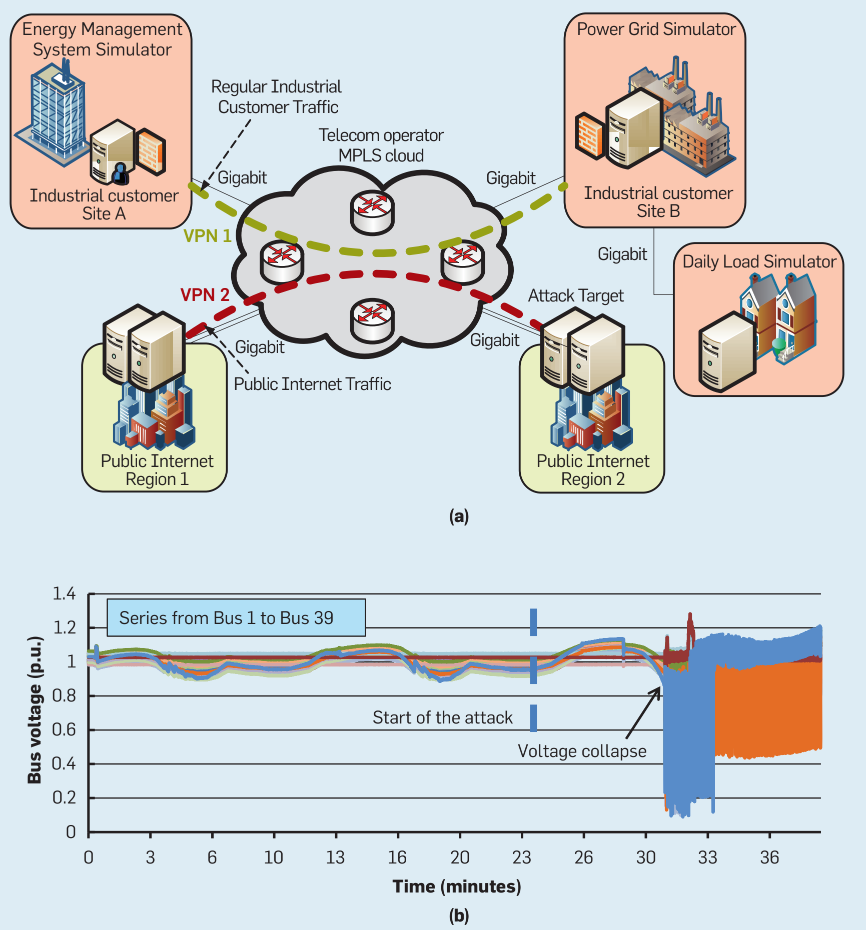
Conjuring a realistic communications infrastructure between EMS and the power-grid simulator, we assumed the service provider was using a Multi Protocol Label Switching (MPLS) network; telecom operators use the MPLS protocol to replace older implementations based on frame relay and asynchronous transfer mode.[12]

Using our Emulab installation, we created a minimal MPLS network with four Cisco 6503 routers on which we defined two MPLS virtual private networks (VPNs). VPN 1 functioned as a protected virtual circuit between Site A and Site B, an approach telecom operators usually follow to isolate customer traffic. Since telecom operators route diverse traffic (such as public Internet traffic) through the same MPLS cloud, we used VPN 2 to create a virtual circuit between two different "public" regions.

Telecom disruption and propagation to the power grid. We then launched a bandwidth-consumption DDoS attack in VPN 2 and measured its effect on the power-grid operator's virtual circuit in VPN 1. The attack caused harm to the grid operator's private circuit. The EMS consequently lost control of the power grid and was unable to send commands that could restore stability. Once an attack begins the grid is able to run approximately seven minutes without intervention (see Figure 3b). However, after those seven minutes the changes in the daily load require intervention of load-shedding algorithms implemented in the EMS. Since the commands from the EMS cannot reach the emulated control hardware, the voltages in the different segments of the grid inevitably begin to collapse.

Shortly after an attack begins, the model becomes highly unstable, exhibiting large oscillations that are difficult to map to reality. One major limitation of simulation-based studies is researchers can reason only within the model's boundaries. However, voltage collapse is a clear indication of grid instability, possibly forcing operators to rebuild an entire grid. For our EMS-related security study it was enough to verify the attacker could redirect the system outside normal operating limits. If experiments go beyond these limits then researchers must likewise extend the models of physical systems to cover extreme and unstable conditions or extend

Figure 3. Effect of a cyberattack on critical infrastructures: (a) experimental setting, with three physical SSim units, an energy-management system simulator, attacker nodes, and two virtual circuits offered by a telecom operator; VPN 1 = grid operator's private circuit, VPN 2 = public Internet; and (b) effect on voltage stability.



the cyber-physical testbed through real physical devices, assuming it is feasible, economically cost effective, and safe.

A look at reality. Most telecom operators limit the interference between separate VPNs; for example, with deployment of quality of service (QoS) in the MPLS network an attack on the public Internet barely affects the private traffic of other telecom customers. We validated this claim by running our EMS-related experiment after activating QoS with packet prioritization (a feature also used to implement packet prioritization in industrial communications) in the MPLS cloud. The only measurable effect was a slight increase of packet round-trip times (by 1ms–2ms), a tolerable delay if we apply the IEEE 1646-2004 standard for communication delays in substation automation, meaning high-speed messages must be delivered in the 2ms–10ms range.

However, such measures, delivered through policies and regulation, are not compulsory. Our EMS-related experiment demonstrated the severe risk if the measures are not implemented, highlighting the potential effect of ICT disruption on a range of physical systems. Moreover, by designing and conducting experiments based on real incidents we were able to explore a number of what-if scenarios. For example, we investigated a 2004 incident that affected Rome's remotely controlled power grid managed through a public telecommunications network.[4] Communications between remote sites was disrupted due to a broken water pipe flooding the server room of a telecom operator, short-circuiting critical hardware. Power-grid operators were unable to monitor or control the remote site. Fortunately, none of the disturbances was harmful, so the grid was stable. Nevertheless, as shown in our experiments on EPIC, a change in the balance between generated and consumed energy would have serious consequences on the electrical grid. In Rome, with a population of 2.5 million in 2004, it could have caused blackouts throughout the city and affected other critical infrastructure (such as transportation and health care).

Conclusion

Combining an Emulab-based testbed with real-time software simulators, EPIC takes a novel approach to cybersecurity studies involving multiple heterogeneous NCIs. EPIC can be viewed as an instance of a new class of scientific instrument—cyber-physical testbeds—suitable for assessing cyberthreats against physical infrastructures. It supports interesting studies in many interdependent critical infrastructure sectors with heterogeneous systems (such as transportation, chemical manufacturing, and power grids); to explore several, see http://ipsc.jrc.ec.europa.eu/?id=691.

References

1. Bell, R. and Åström, K. *Dynamic Models for Boiler-Turbine Alternator Units: Data Logs and Parameter Estimation for a 160MW Unit. Technical Report TFRT–3192.* Lund Institute of Technology, Lund, Sweden, 1987.
2. Benzel, T., Braden, R., Kim, D., Neuman, C., Joseph, A., Sklower, K., Ostrenga, R., and Schwab, S. Experience with DETER: A testbed for security research. In *Proceedings of the International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (Barcelona, Mar. 1–3). IEEE, New York, 2006, 379–388.
3. Bergman, D.C., Jin, D., Nicol, D.M., and Yardley, T. The virtual power system testbed and inter-testbed integration. In *Proceedings of the Second Conference on Cyber Security Experimentation and Test* (Montreal, Aug. 10–14). USENIX Association, Berkeley, CA, 2009, 5–5.
4. Bobbio, A., Bonanni, G., Ciancamerla, E., Clemente, R., Iacomini, A., Minichino, M., Scarlatti, A., Terruggia, R., and Zendri, E. Unavailability of critical SCADA communication links interconnecting a power grid and a telco network. *Reliability Engineering and System Safety 95*, 12 (Dec. 2010), 1345–1357.
5. Charette, R. IT Hiccups of the week: Southwest Airlines computer failure grounded all flights. *IEEE Spectrum Risk Factor Blog* (June 2013); http://spectrum.ieee.org/riskfactor/computing/it/it-hiccups-of-the-week-southwest-airlines-computer-failure-grounded-all-flights
6. Chen, T. and Abu-Nimeh, S. Lessons from Stuxnet. *Computer 44*, 4 (Apr. 2011), 91–93.
7. Chertov, R., Fahmy, S., and Shro, N.B. Fidelity of network simulation and emulation: A case study of TCP-targeted denial-of-service attacks. *ACM Transactions on Modeling and Computer Simulation 19*, 1 (Jan. 2009), 4:1–4:29.
8. Chunlei, W., Lan, F., and Yiqi, D. A simulation environment for SCADA security analysis and assessment. In *Proceedings of the 2010 International Conference on Measuring Technology and Mechatronics Automation* (Changsha City, China, Mar. 13–14). IEEE, New York, 2010, 342–347.
9. Downs, J. and Vogel, E. A plantwide industrial process control problem. *Computers & Chemical Engineering 17*, 3 (Mar. 1993), 245–255.
10. Duggan, D. *Penetration Testing of Industrial Control Systems. Technical Report SAND2005-2846P.* Sandia National Laboratories, Albuquerque, NM, 2005.
11. Hahn, A., Ashok, A., Sridhar, S., and Govindarasu, M. Cyber-physical security testbeds: Architecture, application, and evaluation for smart grid. *IEEE Transactions on the Smart Grid 4*, 2 (June 2013), 847–855.
12. IBM and Cisco. Cisco and IBM provide high-voltage grid operator with increased reliability and manageability of its telecommunication infrastructure. IBM Case Studies, 2007; https://www.cisco.com/web/partners/pr67/downloads/756/partnership/ibm/success/terna_success_story.pdf
13. Manera, M. and Marzullo, A. Modelling the load curve of aggregate electricity consumption using principal components. *Environmental Modeling Software 20*, 11 (Nov. 2005), 1389–1400.
14. McDonald, M.J., Mulder, J., Richardson, B.T., Cassidy, R.H., Chavez, A., Pattengale, N.D., Pollock, G.M., Urrea, J.M., Schwartz, M.D., Atkins, W.D., and Halbgewachs, R.D. *Modeling and Simulation for Cyber-Physical System Security Research, Development, and Applications. Technical Report SAND2010-0568.* Sandia National Laboratories, Albuquerque, NM, 2010.
15. Nai Fovino, I., Masera, M., Guidi, L., and Carpi, G. An experimental platform for assessing SCADA vulnerabilities and countermeasures in power plants. In *Proceedings of the Third Conference on Human System Interactions* (Rzeszow, Poland, May 13–15). IEEE, New York, 2010, 679–686.
16. Nan, C., Eusgeld, I., and Kröger, W. Analyzing vulnerabilities between SCADA system and SUC due to interdependencies. *Reliability Engineering & System Safety 113* (May 2013), 76–93.
17. Queiroz, C., Mahmood, A., and Tari, Z. SCADASim: A framework for building SCADA simulations. *IEEE Transactions on Smart Grid 2*, 4 (Sept. 2011), 589–597.
18. Ríos, M.A. and Ramos, G. Power system modelling for urban mass-transportation systems. In *Infrastructure Design, Signaling and Security in Railway*. InTech, Rijeka, Croatia, 2012, 179–202.
19. RIPE Network Coordination Centre. *YouTube Hijacking: A RIPE NCC RIS Case Study*, 2008; http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study
20. Siaterlis, C., Garcia, A., and Genge, B. On the use of Emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys and Tutorials 15*, 2 (Second Quarter 2013), 929–942.
21. Tuan, T., Fandino, J., Hadjsaid, N., Sabonnadiere, J., and Vu, H. Emergency load shedding to avoid risks of voltage instability using indicators. *IEEE Transactions on Power Systems 9*, 1 (Feb. 1994), 341–351.
22. University of Washington. *Power Systems Test Case Archive*. Electrical Engineering Department, Seattle, 2012; http://www.ee.washington.edu/research/pstca/
23. U.S. Department of Energy. *National SCADA Test Bed.* Washington, D.C., 2009; http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/NSTB_Fact_Sheet_FINAL_09-16-09.pdf
24. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, Dec. 9–11). USENIX Association, Berkeley, CA, 2002, 255–270.
25. Yardley, T., Berthier, R., Nicol, D., and Sanders, W. Smart grid protocol testing through cyber-physical testbeds. In *Proceedings of the Fourth IEEE PES Innovative Smart Grid Technologies Conference* (Washington, D.C., Feb. 24–27). IEEE Power and Energy Society, NJ, 2013, 1–6.

Christos Siaterlis (christos.siaterlis@jrc.ec.europa.eu) is a project officer in the Institute for the Protection and Security of the Citizen of the European Commission's Joint Research Centre, Ispra, Italy.

Béla Genge (bela.genge@ing.upm.ro) is a Marie Curie postdoctoral fellow and a member of the Informatics Department at Petru Maior University of Tîrgu Mureş, Mureş, Romania.

DOI:10.1145/2602574

How to use, and influence, consumer social communications to improve business performance, reputation, and profit.

BY WEIGUO FAN AND MICHAEL D. GORDON

The Power of Social Media Analytics

WITH MORE THAN 4,100 properties in more than 90 countries, Accor Hospitality was facing pressure from customers, as well as from shareholders, to increase customer satisfaction and quality of service during an economic downturn. It thus turned to Synthesio, a global, multilingual social-media monitoring-and-research company, to examine the more than 5,000 customer opinions posted each month on travel websites worldwide concerning Accor's various brands. Accor saw its main challenge as how to quickly identify customer dissatisfaction and then correct problems at their source. Synthesio created a tool to track the online reputations of 12,000 hotels, including those run by Accor and those run by its competitors. It quickly revealed a number of problems Accor guests were having; for example, room keys were being demagnetized unintentionally by their smartphones. Accor was then able to work with its room-key supplier to address the problem. Accor was also able to set up a rewards-and-training program that encouraged its individual hotels to connect with their guests through online conversations. Within one year of contracting Synthesio, the Novotel brand within the Accor group saw 55% growth in positive feedback in online posts, along with a significant decrease in the number of negative comments.

Social media analytics "is concerned with developing and evaluating informatics tools and frameworks to collect, monitor, analyze, summarize, and visualize social media data ... to facilit[ate] conversations and interactions ... to extract useful patterns and intelligence..."[28] Accor illustrates how social media analytics can help businesses improve their reputations and resulting business performance. Ubiquitous smartphones and other mobile devices, Facebook and YouTube channels devoted to companies and products, and hashtags that make it easy to share experiences instantly combine to create a social media landscape that is quickly becoming part of the fabric of everyday business operations. As the number of users on social media sites continues to grow, so does the need for businesses to monitor and use them to their advantage.

Through the rest of this article, we explore how social media popularity necessitates use of social-media analytics, the underlying stages of the analytics process, the most common social media analytic techniques, and the ways analytics creates business value.

» key insights

- Social media analytics involves a three-stage process: capture, understand, and present.
- Key techniques go beyond text analytics to include opinion mining, sentiment analysis, topic modeling, social network analysis, trend analysis, and visual analytics.
- Businesses can use them to realize value in all phases of a product or service life cycle, including insight into changing consumer interests and tastes, influential users, ad-campaign effectiveness, how to respond to crises, and competitive intelligence.

IMAGE COLLAGE FROM SHUTTERSTOCK.COM

The Need

In the early days of social media—the mid-2000s—PR agencies would monitor customers' posts on a business's own website to try to identify and manage unhappy customers. With the explosion in the number of social media sites and volume of users on them, monitoring alone is not enough to render a complete picture of how a company is doing. Consider the pervasiveness of social media[a]:

- Social networking is the most popular online activity;
- 91% of adults online are regular users of social media; and
- Facebook, YouTube, and Twitter are the second, third, and eighth most-trafficked sites on the Internet, as of April 2014.

However, even these statistics do not fully account for the influence social media has on our lives. Users spend more than 20% of their time online on social media sites. Facebook alone has a worldwide market penetration rate over 12% of the entire online population; in North America it is 50%. These rates are growing quickly, with Facebook alone gaining 170 million new users between the first quarter of 2011 and the first quarter of 2012, an increase of 25%. Facebook mobile use is growing even more quickly, at a 67% annual clip, as of Summer 2013.

The amount of information seen by all these users on a typical day gives a clearer indication of the enormous influence of social media. As of October 2012, Facebook's nearly one billion active users collectively were spending approximately 20,000 years online each day. In the same period, YouTube reported more than one billion views, or 500 years of video (spread among 800 million unique users), and 140 million active Twitter users sent more than 340 million tweets.

These are not simply passive uses. YouTube's analysis of its videos indicates 100 million people take some sort of "social action" each week, by, say, liking, disliking, or commenting on what they see. These actions doubled from 2012 to 2013. Facebook integrates social actions in its online ads today by, for instance, allowing users to see if their friends have liked or voted on products being advertised. Likewise, hashtags on Twitter (as well as other social-media platforms) give users another quick and easy way to express their likes, dislikes, interests, and concerns, presenting further opportunities (or challenges) to businesses striving to use them.

The Process

Social media analytics involves a three-stage process: "capture," "understand," and "present" (see Figure 1), the CUP framework. Capture involves obtaining relevant social media data by monitoring, or "listening," to various social-media sources, archiving relevant data and extracting pertinent information. It can be done by a company itself or through a third-party vendor. Not all captured data is useful, however. Understand selects relevant data for modeling while removing noisy low-quality data, using various advanced data analytic methods on the data and gain insight from them. And present deals with displaying findings from the understand state in a mean-

[a] Throughout this article, we cite statistics from a number of websites that closely track these issues, including http://www.adweek.com, http://www.alexa.com, http://www.internetworldstats.com, and http://www.comscore.com, as well as from social media sites themselves.

ingful way. We derived the CUP framework from familiar, broadly applied input-process-output models, making it consistent with Zeng et al.,[28] whose monitor-and-analyze activities were subsumed by our understand stage and whose summarize-and-visualize activities fall within our present stage.

There is also some overlap among the stages; for instance, the understand stage creates models that can help in the capture stage. Likewise, visual analytics supports human judgments that complement the understand stage, as well as assist in the present stage. The stages are conducted in an ongoing, iterative manner rather than strictly linearly. If the models in the understand stage fail to reveal useful patterns, they may be improved by capturing additional data to increase their predictive power. Likewise, if presented results are not interesting or lack predictive power, it may be necessary to return to the understand or capture stages to adjust the data or tune the parameters used in the analytics. A system supporting social media analytics may go through several iterations before being truly useful. Data analysts and statisticians help develop and test systems before others use them.

Capture. For a business using social media analytics, the capture stage helps identify conversations on social media platforms related to its activities and interests. This is done by collecting enormous amounts of relevant data across hundreds or thousands of social media sources using news feeds and APIs or through crawling. The capture phase covers popular platforms (such as Facebook, Foursquare, Google+, LinkedIn, Pinterest, Twitter, Tumblr, and YouTube), as well as smaller, more specialized sources (such as Internet forums, blogs, microblogs, wikis, news sites, picture-sharing sites, podcasts, and social-bookmarking sites). An enormous amount of data is archived and available to meet businesses' needs. To prepare a dataset for the understand stage, various preprocessing steps may be performed, including data modeling, data and record linking from different sources, stemming, part-of-speech tagging, feature extraction, and other syntactic and semantic operations that support analysis. Information about businesses, users, and events, as well as user comments and feedback and other information, is also extracted for later analytical modeling and analysis.

Being tuned in to changing customer tastes and behavior, businesses can anticipate significant changes in demand and adjust accordingly by ramping production up or down.

The capture stage must balance the need to find information from all quarters (inclusivity) with a focus on sources that are most relevant and authoritative (exclusivity) to assist in more refined understanding.

Understand. When a business collects the conversations related to its products and operations, it must then assess their meaning and generate metrics useful for decision making—the understand stage. Since the capture stage gathers data from many users and sources, a sizeable portion may be noisy and thus have to be removed prior to meaningful analysis. Simple, rule-based text classifiers or more sophisticated classifiers trained on labeled data may be used for this cleaning function. Assessing meaning from the cleaned data can involve statistical methods and other techniques derived from text and data mining, natural language processing, machine translation, and network analysis.[9] The understand stage provides information about user sentiment—how customers feel about a business and its products—and their behavior, including the likelihood of, say, purchasing in response to an ad campaign. Many useful metrics and trends about users can be produced in this stage, covering their backgrounds, interests, concerns, and networks of relationships.

Note the understand stage is the core of the entire social media analytics process. Its results will have a significant effect on the information and metrics in the present stage, thus the success of future decisions or actions a business might take. Depending on techniques used and information sought, certain analyses may be preprocessed offline while others are computed on the fly using data structures optimized for anticipated ad hoc uses. Analysts and business managers may participate directly in the understand stage when visual analytics allows them to see various types and representations of data at once or create visual "slices" that make patterns more apparent.

Present. In this last stage, the results from different analytics are summarized, evaluated, and shown to users in an easy-to-understand format. Visualization techniques may be used to present useful information; one commonly used interface design is the visual dashboard, which aggregates and displays information from multiple sources. Sophisticated visual analytics go beyond the simple display of information. By supporting customized views for different users, they help make sense of large amounts of information, including patterns that are more apparent to people than to machines. Data analysts and statisticians may add extra support.

Key Techniques

Social media analytics encompasses a variety of modeling and analytical techniques from different fields. Here, we highlight the most instrumental in understanding, analyzing, and presenting large amounts of social media data. Some techniques support several stages of social media analytics: Sentiment analysis and trend analysis primarily support the understand stage; topic modeling and social network analysis have primarily application in the understand stage but can support the capture and present stages as well; and visual analytics spans the understand and the present stages.

Opinion mining, or sentiment analysis, is the core technique behind many social media monitoring systems and trend-analysis applications.[b] It leverages computational linguistics, natural language processing, and other methods of text analytics to automatically extract user sentiment or opinions from text sources at any level of granularity (words or phrases, up to entire documents). Such subjective information extracted about people, products, and services supports predicting the movement of stock markets, identifying market trends, analyzing product defects, and managing crises. Relatively simple methods for sentiment analysis include word (phrase) counts (the more a product is mentioned, the more it is assumed to be liked); "polarity lexicons," or lists of positive and negative terms that can be counted when used, as in, say, text messages that mention a product by name;[11] and semantic methods that may compute lexical "distances" between a product's name and each of two opposing terms (such as "poor" and "excellent") to determine sentiment.[25] More complicated approaches distinguish the sentiments about more than one item referenced in the same text item (such as a sentence, paragraph, or text message).[10]

All told, both sophisticated and simple methods of sentiment analysis can be effective or flawed, though most research involving texts, tweets, and other short messages involves simple techniques. Though sentiment analysis is increasingly common, sampling bias in the data can skew results—even if large data samples are confused with unbiased samples—especially in situations where satisfied customers are silent while those with more extreme positions loudly voice their opinions.

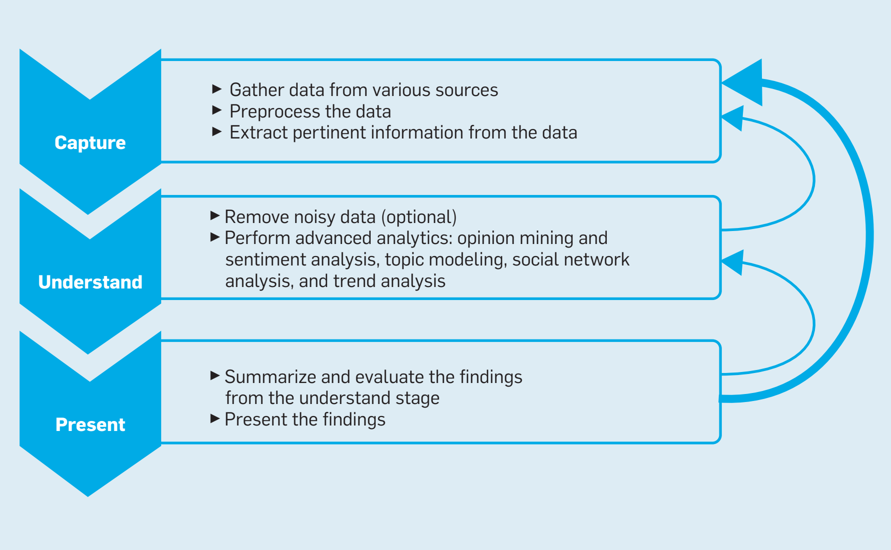
Topic modeling is used to sift through large bodies of captured text to detect dominant themes or topics. Themes can be used to provide consistent labels to explore the text collection or build effective navigational interfaces. Themes can also be used to feed other analytical tasks (such as discovering user interests, detecting emerging topics in forums or social media postings, and summarizing parts, or all, of a text collection). Recent advances in topic modeling also allow these algorithms to be used with streaming data from Twitter and other continuous data feeds, making the technique an increasingly important analytic tool.

Topic modeling uses a variety of advanced statistics and machine-learning techniques; for instance, a number of models identify "latent" topics through the co-occurrence frequencies of words within a single communication[14] or between topics and communities of users.[27] Information about the position of words within messages can also be considered;[26] see Blei[4] for a survey of topic modeling.

Social network analysis is used on a social network graph to understand its underlying structure, connections, and theoretical properties, as well as to identify the relative importance of different nodes within the network. A social network graph consists of nodes (users) and associated relationships (edges). Relationships are typically detected through user actions connecting two people directly (such as accepting another user as a "friend"), though they may be inferred from indirect behaviors creating relationships (such as voting, tagging, and commenting).

Social network analysis is used to model social network dynamics and growth (using such features as network density and locations of new node attachments) that help monitor business activity. Social network analysis is the primary technique for identifying key influencers in viral market-

Figure 1. Social media analytics process.


Capture
▸ Gather data from various sources
▸ Preprocess the data
▸ Extract pertinent information from the data
Understand
▸ Remove noisy data (optional)
▸ Perform advanced analytics: opinion mining and sentiment analysis, topic modeling, social network analysis, and trend analysis
Present
▸ Summarize and evaluate the findings from the understand stage
▸ Present the findings

[b] We adopt the view of Pang and Lee[15] who described the terms "opinion mining" and "sentiment analysis" as having multiple definitions, using them broadly and interchangeably to cover the subjective, textual evaluation of source materials or their features.

ing campaigns on Twitter and other social media platforms. It is also used to detect subcommunities within a larger online community (such as discussion forums), allowing greater precision in tailoring products and marketing materials. It is also useful in predictive modeling, as in marketing campaigns aimed at consumers assumed most likely to buy a particular product.[5]

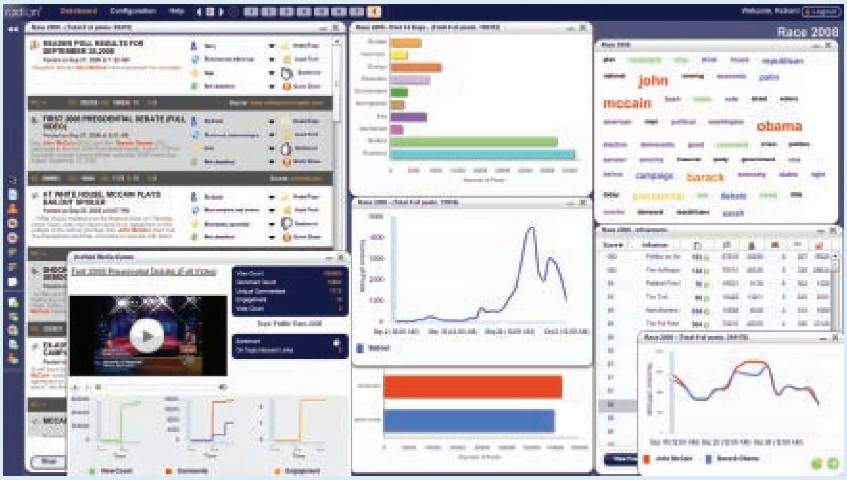
Techniques used by social network analysis to understand the mathematical structure of graphs[18] range from the simple (such as counting the number of edges a node has or computing path lengths) to the sophisticated algorithms that compute eigenvectors (as in Google's PageRank) to determine key nodes in a network. This helps determine whom, say, a business might look to on the basis of expertise and reputation. The analysis of network structure significantly predates the advent of social media, having been developed mainly for analyzing static mathematical graphs. Today's large and continually changing network structures demand new technical approaches, especially when real-time decision support is needed.

Trend analysis is used to predict future outcomes and behaviors based on historical data collected over time. Applications include forecasting the growth of customer or sales numbers, predicting the effectiveness of ad campaigns, staying ahead of shifts in consumer sentiment, and forecasting movement in, say, a stock market. Trend analysis is based on long-standing statistical methods (such as time-series analysis and regression analysis[1]) and other more recent modeling techniques (such as neural networks[12] and support vector machines[20]).

Visual analytics is "the science of analytical reasoning facilitated by interactive visual interfaces."[23] Spurred initially by U.S. defense needs, visualization works across different application areas to support synthesis, exploration, discovery, and confirmation of insight from data that is typically voluminous and spread among different sources. Visual analytics involves a range of activities, from data collection to data-supported decision making. Though many statistical methods underlie visual analytics (such as reducing high-dimensional data to fewer very salient dimensions), the human ability to perceive patterns and draw conclusions is a key factor as well. Indeed, when a torrent of information must be addressed quickly, combining machine and human strengths is critical, in both making decisions and being able to explain and justify them. Visual analytics shares a focus with other visualization techniques on creating economical, intuitive displays, but unlike the classical work of Tufte,[24] these displays must support real-time decision making where the stakes can be high.

Visual analytic systems must be able to process data to reveal hidden structure and detail. Computational methods for data reduction, displaying correlations among disparate data sources, and letting users physically manipulate data displays all underlie visual analytics. Taking a more user-perceptual view, visual analytics can be understood as a collection of techniques that use graphical interfaces to present summarized, heterogeneous information that helps users visually inspect and understand the results of underlying computational processes. One commonly used interface design is the dashboard, where multiple metrics and key performance indicators are portrayed in a way that mimics a car's dashboard design (see Figure 2). Displays typically allow users to interactively interrogate the underlying data and perform data transformations using sliders or other types of controls. Crisis management and detecting breaking events from social media chatter can both benefit from visual analytics. The challenge for visual analytics is to remain responsive to, and create better visual representations for, increasingly massive and complex data requiring speedier interpretation and display on large numbers of devices, from handhelds to full-wall displays.

Figure 2. Radian6 analysis dashboard.



Business Value

Here, we consider the techniques behind social media analytics in more detail, adopting a life-cycle analysis framework. Social media has changed our conversations about products and services but not about the business activities behind them. A life-cycle analysis considers the life of a product (or service) from design to disposal, as well as the supporting activities that take place in parallel with these activities. Although various authors describe the product life cycle with different levels of granularity, one that is quite typical suffices for our purpose, having four stages: design-development; production; utilization; and disposal.[2] Social media is most relevant to the design-development and utilization stages; in addition, social media analytics helps businesses gather competitive intelligence and understand more completely their business environments, suppliers, and competitors. Our use of a life-cycle framework is consistent with other social media analyses.[5]

Product design-development. This first life-cycle stage covers the conceptual, preliminary, and detailed design of a product during which a variety of risks threaten success.[3] Risks involving technology change may be due to misjudging the gaps in technology among different products or from time-to-market pressures. Risks involving design may be due to poor selection of product features, improper differentiation with other products, lack of modularity, or reliance on the wrong parts.

Trend analysis and other social media analytic tools help identify any changes in taste, behavior, and sentiment affecting product design and development. They let designers add and adjust features and help create sufficient lead time for creating next-generation products or even products in a completely new category. They also promote product innovation by capturing and understanding conversations involving either of two groups: loyal customers and average customers. On the one hand, a business's most loyal customers can reveal important insights, as Del Monte Foods, Inc. found in creating and launching a new dog-food product in just six weeks. On the other hand, conversations with "average" customers can also lead to product improvements; for instance, Dell Inc. created its IdeaStorm website in 2007 to solicit users' ideas about improving its computer products and services. Dell takes these suggestions seriously, soliciting comments from others as (dis)confirmation and making changes to its products as needed.

The software industry has taken the lead in social media-based product testing (leading to changes to software) by releasing various versions of its products and soliciting reactions and, in the case of open source programs, allowing user changes. Other industries have followed suit. The most advanced use of social media-based conversations is in the "co-creation" of products, where online users and businesses function as informal partners in generating new product ideas and even entirely new product categories.[16]

Product production. During product production, social media analytics can mitigate risk involving supply-chain responsiveness.[3] Being tuned in to changing customer tastes and behavior, businesses can anticipate significant changes in demand and adjust accordingly by ramping production up or down. Visual analytics can be useful in identifying correlations, outliers, geographic patterns, and other trends that support smoother functioning. A business can also use social media analytics to learn another business with which it competes (or perhaps does not) is having trouble with a supplier, information useful in anticipating and avoiding the same problem. Close monitoring of social media can even help in technical-administrative tasks. For instance, inventory management is based on forecasts and production schedules. Social media can give advance warning when situations that might affect the acquisition of resources become less predictable, including political tensions overseas that could disrupt the flow of metals, minerals, and other vital supplies for manufacturing.

Over 50% of all online users expect a response to a complaint the same day they send it, though less than one-third receive one.

Product utilization. The most common use of social media analytics is in the product utilization life-cycle stage and involves three key social media objectives: brand awareness, brand engagement, and word of mouth.[13] Brand awareness introduces customers to a brand (or product) or increases customers' familiarity with the brand. Brand engagement increases customers' connections with a brand. Word of mouth encourages users' efforts to positively influence other users' purchasing behavior.

A number of metrics have been proposed for assessing social media effectiveness in this stage;[13] for example, for microblogging platforms like Twitter, simple ones include number of tweets and followers (for brand awareness), number of followers and replies (for brand engagement), and number of retweets (for word of mouth). Although they provide important information, they are no substitute for more powerful techniques in social media. For instance, influencer profiling uses social media to develop a deep understanding of users' backgrounds, tastes, and buying behavior to create better customer segmentation. Segmentation assists businesses in reaching various groups, using the differences to guide different strategies for increasing brand awareness and engagement for each group. Influencer profiling also assists in identifying social-community leaders or experts whose opinions are valuable in product development and consumer-supported customer service. Techniques for influencer profiling include social network analysis, topic modeling, and visual analytics.

Brand engagement suggests consumers feel a personal connection to a brand. Psychometric constructs suggesting brand engagement include the terms "special bond," "identify with," and "part of myself."[19] Trying to create such relationships, businesses create a variety of activities, including "liking" and "commenting" on product websites. Other activities aim to generate a deeper sense of connection, often by enticing playful user actions. For instance, the German car manufacturer Audi AG was the first to use a hashtag in its 2011 Super Bowl ad, showing partying, good-looking vampires, and concluding with #SoLongVampires. This memorable hashtag could be tweeted during the most-watched sporting event of the season. More important, Audi then used social media tools to follow users who tweeted the hashtag to initiate a real-time dialogue that was one step among many in cultivating relationships with potential new customers. To the benefit of Audi and its brand, by the end of the Super Bowl, the hashtag had become a trending topic on Twitter.

More broadly, social media analytics allows businesses to judge online reactions to ad campaigns. The resulting metrics help link a campaign to subsequent sales, and thus the success of the campaign. Customers' reactions can also help alter the campaign in accordance with their likes and dislikes.

Word of mouth extends consumers' engagement beyond interactions with products to other consumers. Businesses hope these interactions, through, say, retweets, reblogs, and social tagging, are positive, though it does not always mean they are.

Customers' online complaints about products and services are common, with, for instance, nearly two-thirds of all customers worldwide using social media for this purpose. Over 50% of all online users expect a response to a complaint the same day they send it, though less than one-third receive one. Most top-50 brands worldwide never respond to customer comments on their own websites, hurting brand image and reputations.[17] The viral spread of user complaints through social media can hurt business significantly.

Unlike gathering business intelligence from other sources, obtaining information from social media about suppliers or competitors (in all they do) is almost as easy for a business as monitoring its own affairs.

Real-time sentiment analysis, topic modeling, and other tools allow businesses to know how their customers feel about their products and services and respond quickly, before customer complaints become an online torrent. Trends in Twitter and other informal data generated during the cholera outbreak in Haiti following the 2010 earthquake were significantly correlated with official data used by the Haitian government and the World Health Organization to respond to the epidemic but were available two weeks earlier.[8] Similarly, social media data provides early warnings that, ignored, can create impressions of a business that are difficult to overcome.

A 2010 study of 20 brand marketers worldwide showed the top 1% of a website's audience shares up to one-fifth of all links to the site and influences up to one-third of the actions taken by other users.[22] Social network analysis can be used to determine who these key users are so they stay satisfied, engaged, and help a business market its products on its own website and via word of mouth through these users' social networks.

Product disposal. Nearing the end of a product's life span, consumers may have to decide how they will dispose of and replace it. For many consumers, being able to responsibly (ecologically) dispose of a product (such as a computer) may influence their overall impression of a company and its products. Creating convenient, responsible disposal and ensuring consumers are aware of it is important to the company, as well as to the environment. Social media analytics can be used to track consumer concerns, and companies themselves can engage in online conversations about disposal. Savvy companies that track these conversations can also infer that disposal may be accompanied by the purchase of a replacement item and leverage that knowledge in their marketing.

Competitive intelligence. Our discussion of the business value of social media analytics through the life-cycle framework has so far focused on products, services, and customers. But social media analytics also provides "competitive intelligence" by helping businesses understand their environments, suppliers, competitors, and overall business trends. Unlike gathering business intelligence from other sources, obtaining information from social media about suppliers or competitors (in all they do) is almost as easy for a business as monitoring its own affairs.

Social media analytics can also play a key role in helping identify and respond to crises. Ironically, businesses often cause crises through their own efforts to disseminate messages through social media. Large organizations (such as the American Red Cross, Burger King, and Chapstick) have been implicated in social media messages that were ill-received, even when, as in the case of the Red Cross in 2011, disseminated unintentionally. The Red Cross and Burger King quickly acknowledged their mistakes and took action, deftly, in the case of the Red Cross, defusing a potential crisis, first by responding with humor, then, after identifying the uncommon hashtag in the inadvertent message sent from its account, using it to generate a successful blood-donation campaign. Unlike the Red Cross and Burger King, Chapstick at first failed to respond to customers' complaints at all, then removed them from its website without responding. These actions exacerbated the bad publicity it generated from its online presence.

Conclusion

Even as some social media websites explode into use, quickly becoming everyday tools (Facebook launched in 2004, Twitter in 2006), new platforms are constantly emerging; for example, Pinterest launched in Summer 2011 and, as of June 2013, had approximately 50 million users worldwide. All told, a dozen sites, also as of June 2013, had at least 100,000 registered users and many more unique visitors, according to Wikipedia, including sites (such as Ozone and Sina Weibo) most of the online population has never heard of.

Even as businesses begin to recognize the business risk of ignoring social media content and conversely its inherent opportunity, their questions suggest how much remains unknown; for example, a 2012 survey of 3,800 marketers in the U.S. indicated three top concerns:[21]

- ▸ How to track social media return on investment;
- ▸ How to identify and engage with the most influential social media users; and
- ▸ What tactics to use to create an effective social media strategy.

Social media analytical tools are designed to address each of them. At the same time, social media transform the very nature of business. Current patterns suggest social media could produce an additional $940 billion in annual consumption, especially in electronics, hardware, software, and mobile technologies.[7] Social media support "co-creation" of products, with consumers working online with companies' product designers.[16]

Technical challenges loom as well; for example, the extraordinary volume of big data already challenges social media analytics,[6] and languages add further complications, as businesses monitor and analyze social media conversations around the world. These challenges could swell, as social media analytics begin to incorporate user-based location data facilitated by mobile technology, and consumer pressure increases to process and respond to social messages in real time.

Acknowledgments

We thank Jonathan Woody of Virginia Tech for his data-collection support related to various parts of this article. We also thank the anonymous reviewers for their valuable comments in helping us improve the article.

References

1. Anderson, T.W. *The Statistical Analysis of Time Series*. John Wiley & Sons, Inc., New York, 1971.
2. Asiedu, Y. and Gu, P. Product life cycle cost analysis: State-of-the-art review. International Journal of Product Research 36, 4 (1998), 883–908.
3. Billington, C., Lee H., and Tang C. Successful strategies for product rollovers. *Sloan Management Review 39*, 3 (Apr. 2012).
4. Blei, D.M. Probabilistic topic models. *Commun. ACM 55*, 4 (Apr. 2012), 77–84.
5. Bonchi, F., Castillo, C., Gionis, A., and Jaimes, A. Social network analysis and mining for business applications. *ACM Transactions on Intelligent Systems and Technology 2*, 3 (Apr. 2011), 22.
6. Chen, H., Chiang, R.H.L., and Storey, V. Business intelligence and analytics: From big data to big impact. *MIS Quarterly 36*, 4 (Dec. 2012), 1165–1188.
7. Chui, M., Manyika, J., Bughin, J., Dobbs, R., Roxburgh, C., Sarrazin, H, Sands, G., Westergren, M. T*he Social Economy: Unlocking Value and Productivity Through Social Technologies. Research Report.* McKinsey Global Institute, 2012; http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/the_social_economy
8. Chunara, R., Andrews, J.R., and Brownstein, J.S. Social and news media enable estimation of epidemiological patterns early in the 2010 Haitian cholera outbreak. *The American Journal of Tropical Medicine and Hygiene 86*, 1 (Jan. 2012), 39–45.
9. Fan, W., Wallace, L., Rich, S., and Zhang, Z. Tapping the power of text mining. *Commun. ACM 49*, 9 (Sept. 2006), 77–82.
10. Feldman, R. and Sanger, J. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press, Cambridge, U.K., 2007.
11. Gayo-Avello, D. Don't turn social media into another 'Literary Digest' poll. *Commun. ACM 54*, 10 (Oct. 2011).
12. Grossberg, S. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks 1*, 1 (1988), 17–61.
13. Hoffman, D. and Fodor, M. Can you measure the ROI of your social media marketing? *Sloan Management Review 52*, 1 (Fall 2010), 41–49.
14. Hofmann, T. Probabilistic latent semantic indexing. In *Proceedings of the ACM SIGIR International Conference on Research and Development in Information Retrieval* (Berkeley, CA). ACM Press, New York, 1999, 50–57.
15. Pang, B. and Lee, L. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval 2*, 1 (2008), 1–135.
16. Ramaswamy, V. and Gouillart, F. *The Power of Co-Creation.* Free Press, London, 2010.
17. Rollason, H. Why social media makes customer service better. *Mashable* (Sept. 29, 2012); http://mashable.com/2012/09/29/social-media-better-customer-service/
18. Scott, J. *Social Network Analysis.* SAGE Publications, Thousand Oaks, CA, 2012.
19. Sprott, D., Czellar, S., and Spangenberg, E. The importance of a general measure of brand engagement on market behavior: Development and validation of a scale. *Journal of Marketing Research 46*, 1 (Feb. 2009), 92–104.
20. Steinwart, I. and Christmann, A. *Support Vector Machines.* John Wiley & Sons, Inc., New York, 2008.
21. Stelzner, M.A. *Social Media Marketing Industry Report.* 2012; http://www.socialmediaexaminer.com/social-media-marketing-industry-report-2012/
22. Straley, B. How to: Target social media influencers to boost traffic and sales. *Mashable* (Apr. 15, 2010); http://mashable.com/2010/04/15/social-media-influencers/
23. Thomas, J.J. and Cook, K.A. A visual analytics agenda. *IEEE Computer Graphics and Applications 26*, 1 (Jan./Feb. 2006), 10–13.
24. Tufte, E.R. *The Visual Display of Quantitative Information, Second Edition.* Graphics Press, Cheshire, CT, 2001.
25. Turney, P. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the Association for Computational Linguistics* (Philadelphia). ACL, Stroudsburg, PA, 2002, 417–424.
26. Wallach, H.M. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23rd International Conference on Machine Learning* (Pittsburgh). ACM Press, New York, 2006, 977–984.
27. Yin, Z., Cao, L., Gu, Q., and Han, J. Latent community topic analysis: Integration of community discovery with topic modeling. *ACM Transactions on Intelligent Systems and Technology 3*, 4 (Sept. 2012), 1–21.
28. Zeng, D., Chen, H., Lusch, R., and Li, S-H. Social media analytics and intelligence. *IEEE Intelligent Systems 25*, 6 (Dec. 2010), 13–16.

Weiguo Fan (wfan@vt.edu) is a distinguished guest professor at Xi'an Jiaotong University, Xi'an, China, and a professor of accounting and information systems and director of the Center for Business Intelligence and Analytics in the Pamplin College of Business at Virginia Tech, Blacksburg, VA.

Michael D. Gordon (mdgordon@umich.edu) is the Arthur F. Thurnau Professor of Entrepreneurial Studies at the Ross School of Business of the University of Michigan, Ann Arbor, MI.

DOI:10.1145/2602695.2602701

These interactive applications let users perform, and thus preserve, traditional culture-defining crafts.

BY DANIELA ROSNER, MARCO ROCCETTI, AND GUSTAVO MARFIA

The Digitization of Cultural Practices

OVER THE PAST few decades, computing researchers have developed a wealth of new systems to preserve and share cultural resources. Computing platforms give everyone access to high-resolution images of ancient cave paintings and Google artwork; 3D scanning systems digitize Michelangelo's sculptures and Pompeii's remains;[7] and haptic devices allow users to touch and feel ancient sculptural forms.[1] More recently, researchers have employed collaborative and crowd-sourced software in the restoration of specific artifacts and environments (such as paintings and archeological sites).[9] These approaches tend to fall within one of three categories: digitally reconstructing objects and landscapes from the past; broadening access to cultural resources through remote distribution platforms; and digitally representing and archiving cultural artifacts and media (see Figure 1). In each, researchers position digitization techniques to support cultural preservation.

However, among these technological developments, researchers often miss a key element—cultural practices. Even as our cultural artifacts and media carry on digitally for future generations, our reasons for reading books, exploring contexts for understanding artwork, and developing ways to share and celebrate everyday practices remain relatively rare. And for good reason, as computing researchers find it difficult to identify, codify, and digitally record cultural practices. Cultural practices do not figure as squarely in the language of computation as many abstract and theoretical models. They emerge as forms of memory, in which the public remembers and forgets the past as a lived and constantly mutating collective experience. In this regard, researchers do not conserve cultural practices in the sense they freeze them in time by making them explicit. On the contrary, they preserve cultural practices by "enacting" tradition and experience. Digitization can thus enable and extend the work of preservation. In making this argument, we look at practices, after Schatzki,[15] as arrays of human activity, temporally situated events involving rehearsed, materi-

» key insights

- Even as they contribute to the process of digitizing documentary collections, technologies involving cultural heritage often ignore cultural practices.
- A new kind of research aims to identify how established sociocultural practices shape and are shaped by digital technologies.
- Technologies for digital preservation can help frame the connections between tradition and progress in a cultural sense yet fail to connect objects to their history and to the people who created them.

From Artist's View iPad app for exhibition "Americans in Florence. Sargent and the American Impressionists" 2012. Netribe, Bologna, Italy.

PHOTOS COURTESY OF PALAZZOSTROZZI.ORG

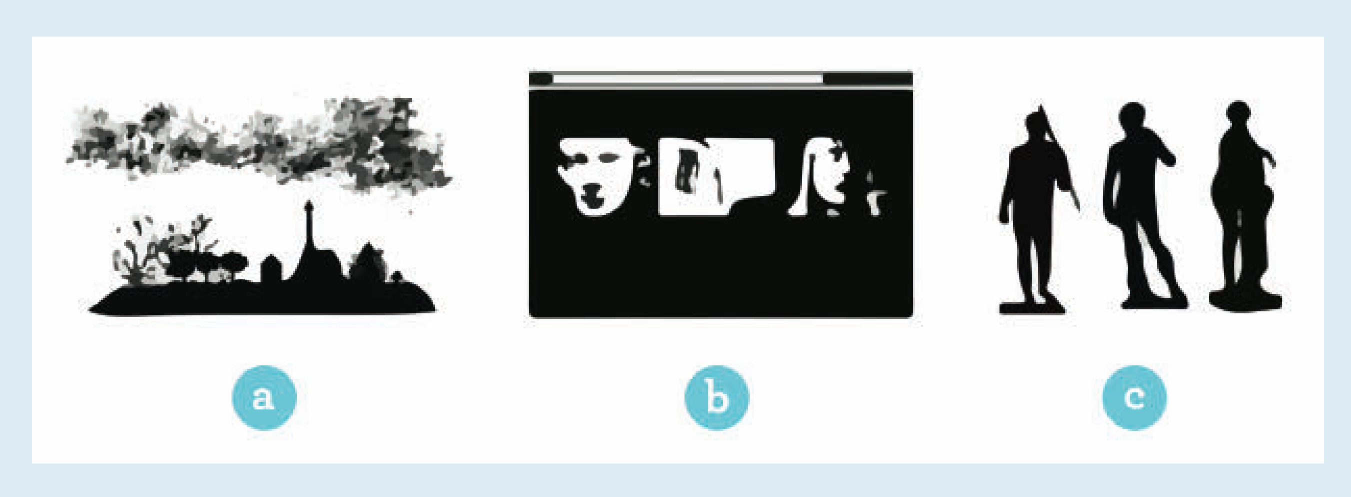
ally mediated actions and embodied social relations. The United Nations Educational, Scientific, and Cultural Organization (http://en.unesco.org/) identifies practices as "very fragile by their very nature." While they bring to life some of our most celebrated cultural forms, they are easily overlooked by the public.[2]

Here, we consider the role computing techniques play in sustaining cultural practices. By tracing them in relation to developments in computing research, we see a need to revise our widely accepted digital preservation techniques. Inspired by a large body of literature from the digital humanities, we urge understanding the dynamic nature of cultural forms as an important strategy for preservation. However, this view does not fit within the dominant conceptual framework defined by computing researchers, a rubric grounded in the assumption that cultural histories are stabilized by the material forms on which they are encoded, stored, and replicated.

As an example of how culture might be preserved, consider the Artist's View iPad application,[11] designed by Netribe of the Palazzo Strozzi in Florence, Italy, to familiarize museum visitors with the paintings in the 2012 exhibition "Americans in Florence: Sargent and the American Impressionists." Its designers took advantage of the museum's close proximity to many of the Florentine and Tuscan landscapes portrayed in the paintings by guiding visitors through the locations in which exhibiting artists painted or found their inspiration. When the visitors would reach any of the locations, the application would let them juxtapose the painting with the scene the painting represents. The application does more than reconstruct objects from the past by leveraging geolocation and augmented reality techniques so users imagine the spark behind a given artist's inspiration. Such applications give people a rich context with which to interpret and engage historic artifacts, providing access to aesthetic resources and uncanny moments hidden from view on a traditional museum floor. By extending the practice of visiting, paintings, books, and other historical artifacts could become part of a visitor's deeper museum experience.

The Palazzo Strozzi thus put in place an intriguing though limited interaction with cultural history. We observed the same ecological environments that once inspired the creators of Renaissance paintings, yet our view was shaped by the modern contexts in which they develop. Visitors today can still discover Frank Duveneck's tranquil rendering of the Ponte Vecchio bridge as they walk along contemporary streets, full of bustling commerce and international tourism. The painterly depictions of landscapes are at once drawn near and given depth. Situated in the same space at another time, they feel physically familiar but socially out of reach. Sometimes evocative, other times pedagogical, the software renderings share a specific (and partial) view of an artist's brushwork and composition.

Figure 1. Systems designed to sustain cultural resources: (a) digitally reconstructing objects and landscapes from the past; (b) broadening access to cultural resources through remote distribution platforms; and (c) digitally representing and archiving cultural artifacts and media.



Combining modern computer-vision techniques with historical objects, applications like Artist's View prompt questions like: What can computing do to facilitate our experience with the past? What are the limits of cultural engagement through such modern technological developments as gesture recognition, networking, and miniaturized displays? And how can computing techniques inform what it means to preserve cultural practices alongside artifacts? To answer, consider two projects that begin to extend and disrupt today's work of computational preservation.

Exploring Cultural Practices with Computation

One author, Roccetti, discovered his allergy to gluten when he was in his 40s, resulting in disconnection with the culinary aspect of his Italian heritage, traditionally linked to such food. Though not particularly unusual, Roccetti's allergy introduced a challenge, as he could no longer consume pasta, a staple of Italian culinary life. As a lifelong resident of Bologna, the gastronomically (and politically) vibrant city south of Venice, Roccetti sought to connect with a celebrated aspect of his past. Bolognese tortellini, the small belly-button-shaped dumplings traditionally prepared in chicken broth, had pervaded his childhood and that of generations of family members before him. Working with another author, Marfia, as well as with others at the University of Bologna, Roccetti viewed exploring tortellini as not only a delicacy to consume but as a cultural pastime worth preserving—the technique of producing pasta. Out of this interest came Tortellino X-perience, a game designed to teach people how to make the Northern Italian tortellini (see Figure 2).[13] Designed for the 2010 International Expo of Shanghai, the system uses a webcam and simple gesture-recognition techniques to enable Expo visitors, as well as users today, to learn and repeat the basic methods required to make tortellini. Projecting an interactive video above an empty countertop, the system prompts them to repeat a set of prespecified actions in preparation of a tortellini recipe, parsing the actions in the form of a game. Particular actions are required during each preparation phase and shown to the player. When the player reproduces the actions in the video, a set of vision algorithms compares the player's gestures against a set of encoded demonstrated actions. Finally, the system presents the subsequent step in the tortellini-making process.

In practice, the system leads users through a step-by-step process to perform all actions necessary to prepare tortellini. In the first step, users pour flour, eggs, and water onto the center of the countertop. In the second, they mix together all the ingredients, making circular gestures, until a ball of dough takes shape. They perform the third step with a rolling pin, rolling it forward and backward until a thin layer of dough takes shape. In the fourth step, they cut the thin foil of dough into squares with a knife. The fifth step involves stuffing the resulting squares, and the sixth and seventh folding the stuffed squares of dough, finally ready to be cooked. Et voilà, virtual tortellini are ready for virtual consumption.

Thousands of miles west of Bologna, the third author, Rosner, had been designing interactive media for the Adler Planetarium and Astronomy Museum in Chicago. In collaboration with educators and astronomers, she used Adobe Macromedia software to develop interactive games for exploring astrophysical data. The software was made available to museum visitors through kiosks throughout the museum's exhibit halls. Producing and maintaining the software—and resulting science narratives—involved installing frequent software updates, fixing broken hardware, and monitoring networking errors. This design work was unlike the creative activities (such as knitting) she had practiced since childhood. Knitting, as a process of interlocking loops of yarn with two needles, presented tangible histories of production by indexing her handiwork in the stitches. While Rosner worked at the Adler, a growing number of people was rediscovering or newly embracing the activity of knitting in which they discovered disruptive politico-spatial practices (such as "yarn bombing"), or applying old knitted garments like scarves and sweaters to public infrastructure (such as lamp poles, fire hydrants, and trees). She also discovered how to add digital enhancements to knitting; knitters regularly use online videos and how-to instructions to learn new techniques, connecting with other knitters on craft-related social-networking sites and posting updates on knitting blogs. Rosner was struck by the qualitative difference between the vibrant social relationships emerging around tools for knitting and the interactive digital displays she had designed for the museum environment. Knitting seemed to embody cultural practice that contrasted with conventional digital content presentation and codification tools. Knitting emphasized the means of production alongside the artifacts being produced.

With knitting as a focus, Rosner began work with Kimiko Ryokai of the University of California, Berkeley, to develop Spyn[14] a system that enables association of digital records with locations on hand-knit fabric (see Figure 3). The system evolved over three years (2009–2011) from an Adobe Flex application running on a Mobile PC to an Android application running on a T-Mobile G1 smartphone. As they knit, knitters could mark a position on fabric and link digital media (such as images, text, and video) to the same location on the garment. By pointing a smartphone's camera at a desired position, they could later retrieve the associated information. The original system used infrared ink patterns printed on yarn to associate locations on fabric with the collected media. The pattern could be recognized through a simple visual edge-detection algorithm and used as an index, or virtual link, to retrieve the digital content (such as a homemade video or musical sounds) linked to the position. The system still allows knitters to store associations between digital media and fabric through their smartphones.

Tortellino X-perience and Spyn enable people to enact traditional experiences by creating new cultural resources. Tortellino X-perience engages tortellini making by teaching embodied techniques, tacit knowledge, and presuppositions that could, in turn, impart and spur interest in Bolognese culture. Spyn extends opportunities for social annotation and storytelling around the highly feminized handicraft of knitting by allowing particular social relations to emerge.

Taking a step back, both projects inform how we think about the digitization of cultural practices. The kinds of information that become most relevant, useful, and evocative when preparing tortellini or knitting sweaters differed among the people using these tools. Some visitors to the Tortellino X-perience exhibit found the sequence of actions depicted on screen a compelling activity in learning to cook. Others narrowed their focus on the resemblance between their own gestures and those shown on the screen, gaining social value from performing similar gestures. With Spyn, the volume of recorded media associated with the knitted fabric being created overwhelmed some recipients. The pressure for reciprocity could intensify to the point of discomfort, as when one user referred to the Spyn knit as "emotional blackmail." Other times the digital records presented a sense of timelessness, or, as another user called it, a "4D object." The system shifted what knitting could accomplish, enabling the recipient of the garment to access stories of its making, sometimes changing the meaning of the knitted item (such as by turning a pair of gloves into a travel journal, a hat into a mix tape, or a vest into a puzzle). Such varied responses illustrate the fickle nature of the digital, or how social environments in which cultural practices are situated inform and transform how digital information is interpreted and disclosed over time.

These projects also highlight new opportunities for sustaining the kinds of data we might want to preserve. Tortellini apprentices may not care about the precision of gesture replication, as gestures are often readjusted and re-

Figure 2. Tortellino X-perience; storyboard steps.

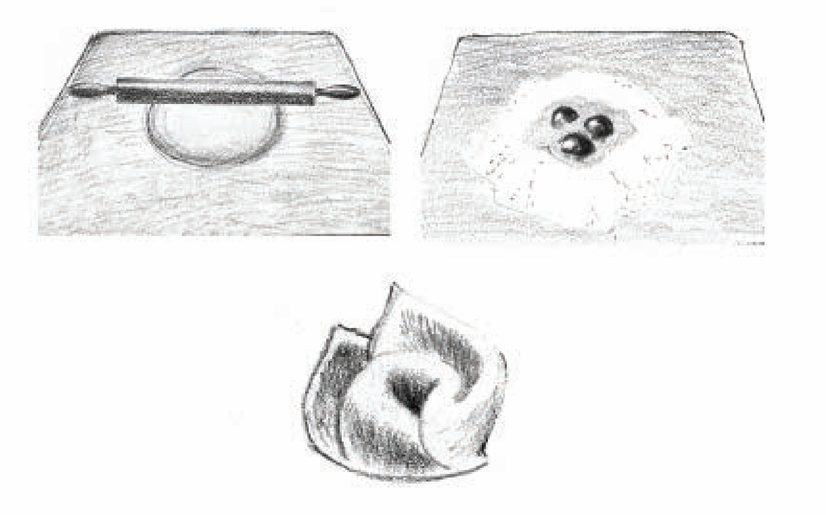
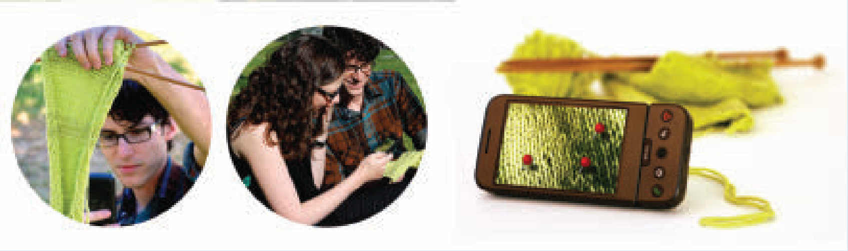


Figure 3. Spyn mobile phone software associates digital records in the creative process—captured through audiovisual media, text, and geographic data—with physical locations on handmade fabric; credit: James Jordan, *California Magazine*.



oriented while working. Knitters may not care how many stitches it took to make a particular hat or what they were thinking when making it, since the hat may already embody some shared value or ideal. In this sense, the data collected through cameras or other sensors has little value in isolation. Rather, the patterns of social activity that unfold as part of engaging the devices—rhythms of interaction expressed through annotated stitches or moments of reflection viewing video sequences—may reveal more about the history and collective memory of a craft practice than would a gesture alone. In enabling people to extend the work of their hands through digital media, the system simultaneously invites the conservation and symbolic transformation of a given practice.

Knitters using Spyn choose the kinds of stories they want to tell through digital media collected while knitting and control the degree to which the fabric is legible to vision algorithms.

Technological Paradox

We thus reach a paradox of preservation: Things change in order to remain the same. As we aim to sustain cultural practices, enabling the "replication" of techniques through digitization, we simultaneously change the replicated forms. Roccetti and Marfia designed Tortellino X-perience to emphasize the technical process, guiding the actions leading to traditional Italian dumplings, yet often neglected the cultural resources and communities of practice in which tortellini making developed. Further, the system separated the means of production from the embodied and tactile engagement particular to its tools—rolling pin, knife, and spoon—and ingredients—flour, eggs, salt, and water. Rosner designed Spyn to focus on the social relations tightly bound with fragments of fabric, perhaps a knitter associating the music listened to while knitting with stitches on her mother's scarf. However, Spyn is not designed to track the pattern or gestures with which the stitches were made. Each system was built to sustain critical modes of cultural engagement but also left some out. Practices of preservation inevitably come with transformation.

Beyond limitations of codification, cultural practices rely on digital infrastructures consisting of hardware and software that could ultimately wear out. As a result, cultural practices not only change but deteriorate as well; for example, with Spyn, Android phones wear out, as the software used to store the data becomes incompatible with subsequent versions of the Android operating system, and users are no longer able to access the data. Preservation yields disruption.

From cave painting to cataloging, humans have always built cultural records of ritual activity and embodied knowledge. By giving their practices material form, they transform performance into imagery, text, and audio to hold onto some aspects of their tacit experience while discarding others. A viewer may not fully grasp the oration reflected in an heirloom painting of a sermon, and the painting may not exist for the next generation if not stored properly. In this sense, the question of what to preserve (and how) is nothing new. Yet with digitization comes something else. Educational institutions can embrace ways of learning and interacting not previously available. As with Tortellino X-perience and Spyn, users perform an activity not only by following the gestures shown in a video but by integrating what they learn from the video with additional practices (such as digital annotation through computer vision). Linking digital forms and patterns, these added resources in turn shift the very purpose of the practice itself; for example, with Spyn, a knitted hat becomes a mixtape. Preservation is not the work of encoding our lived experience but rather enabling memory practices, the modes of performance through which users reimagine the past.

What Next?

A wealth of digitization projects aim to support the capture, storage, and representation of material resources, including books, archeological sites, and works of art;[5] for example, Levoy et al.'s Google Books project[6] designed techniques for scanning fragile texts, making it possible to capture, parse, and store pages from antiquarian books without having to open the books completely. Opening and flattening antiquarian volumes, as with conventional scanners, endangers bookbindings, or glued or sewn bands holding together a book's pages. Injuring pages, in turn, disrupts the process of digitization, as it could degrade what is conventionally viewed as the original artifact, the physical book. However, when establishing a book's provenance depends on the edition of a volume or to whom it belonged, how should scanning be applied? Is the scanner able to capture the mottling on the page margins or evidence of dog-eared pages? Google Books offers a digitization method that applies advanced imaging techniques to digital archiving even as it prompts questions as to what exactly researchers aim to preserve.

In a second project, Proudfoot and Levoy[12] developed what they call a "3D computer archive" of Renaissance artist Michelangelo's sculptures by scanning the original forms as they found them in museums. They sought to create a lasting archive of the artworks by stitching together data from multiple sources, including a planar light field scanner, handheld lightfield scanner, and low-res models for planning purposes. In the process, they accommodated variable ambient light, filling holes through space-carving techniques and aligning scans from multiple gantry angles and positions. This archive increased access to the statues, as well as public availability of the artifacts, even as they displaced their cultural context.

Computing researchers view digitization in these settings as computer science and engineering. Engineers built the scanners and use them to scan and archive physical artifacts. Museum attendees and others using the scans have little influence on what books are available to them and how they were preserved in digital form. While an engineer could decide not to scan the first edition of Miguel de Cervantes Saavedra's *Don Quixote*, believing the fifth and sixth editions were sufficient, a scholar of Latin literature might require the first edition to contextualize 17th century interpretations. In Tortellino X-perience and Spyn, not only engineers but also craftspeople and others—Italian chefs, pasta makers, and knitters—interested in producing the actual items take up the work of digitization. Users of Tortellino X-perience shape the vision algorithm's understanding of events and subsequent sequence of actions shown on a video screen. Knitters using Spyn choose the kinds of stories they want to tell through digital media collected while knitting and control the degree to which the fabric is legible to vision algorithms. By creating, using, and revisiting digitized files, contemporary pasta makers and knitters take part in the digitization and, in turn, the preservation of cultural practices.

Conclusion

Our study of Tortellino X-perience and Spyn find people use computing resources to connect their current experience to past events, comparing their own unique gestures to the delicate pinch of the tortellini dough rendered by a digital display to remember a given technique. Likewise, they use media associated with knitted fabric and garments to recall lost stitches at particular moments. Even though the revisited pinching and knitting techniques will never be exactly as originally performed, the physical dough pinching and needle manipulation help users construct memories and recognize particular practices as part of a cultural tradition. Preserving cultural practices involves moving beyond digital historiography and its institutionalized requirements to explicit evidence, documentation, authenticity, and provenance. Viewing cultural practices through the lens of memory practices, researchers and practitioners find them "preserved" through histories in constant flux, intertwined with collective memory.

Exploring computing technologies through the lens of cultural practices reveals the complex nature of digitization and degradation, enabling us to comment on the status of the ephemeral, how when we trace the ephemeral, the ephemeral changes the behavior we hope to trace. Modern technologies are able to register and reify the practices central to our cultural heritage but also reconfigure them.[4] We thus face the challenge of understanding the kinds of transformations made possible through digitization and the physical craft-making patterns due to cultural preservation.

Acknowledgment

The authors would like to thank the Italian ALTruism Enabling Resource Network for supporting this work.

References

1. Bergamasco, M., Avizzano, C., Di Pietro, G., Barbagli, F., and Frisoli, A. The museum of pure form: System architecture, robot and human interactive communication. In *Proceedings of the 10th IEEE International Workshop on Robot and Human Interactive Communication* (Paris, Sept. 18–21). IEEE Press, Piscataway, NJ, 112–117.
2. Bouchenaki, M. The interdependency of the tangible and intangible cultural heritage. In *Proceedings of the 14th ICOMOS General Assembly and International Symposium* (Victoria Falls, Zimbabwe, Oct. 27–31). ICOMOS, Paris, 2003, 1–5.
3. Brown, J.S. and Duguid P. Borderline issues: Social and material aspects of design. *Human-Computer Interaction 9*, 1 (1994), 3–36.
4. Ferretti, S., Furini, M., Palazzi, C.E., Roccetti, M., and Salomoni, P. WWW recycling for a better world. *Commun. ACM 53*, 4 (Apr. 2010), 139–143.
5. Kallinikos, J., Aaltonen, A., and Marton, A. A theory of digital objects. *First Monday 15*, 6 (2010); http://pear.accc.uic.edu/ojs/index.php/fm/article/view/3033/2564
6. Levoy, M., O'Sullivan, J.K., and Uhlik, C.R. *Acquiring and Using Three-Dimensional Information in a Document Scanning System. U.S. Patent No. 7,586,655.* U.S. Patent and Trademark Office, Washington, D.C., 2009; http://assignments.uspto.gov/assignments/q?db=pat&pat=7586655
7. Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J. and Fulk, D. The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New Orleans, July 23–28). ACM Press, New York, 2000, 131–144.
8. Mayer-Schonberger, V. *Delete: The Virtue of Forgetting in the Digital Age.* Princeton University Press, Princeton, NJ, 2011.
9. Mazzeo, R., Palazzi, C.E., Roccetti, M., and Sciutto, G. Computer-assisted pigment identification in artworks. In *Proceedings of the Third European Conference on Internet and Multimedia Systems and Applications* (Chamonix, France, Mar. 14–16). ACTA Press, Anaheim, CA, 2007, 266–271.
10. McIntosh, A. and Prentice, R.C. Affirming authenticity: Consuming cultural heritage. *Annals of Tourism Research 26*, 3 (July 1999), 589–612.
11. Netribe. Artist's View iPad app. Bologna, Italy; https://itunes.apple.com/mg/app/artists-view/id504132267?mt=8
12. Proudfoot, R.A. and Levoy, M. *Systems and Methods for Glare Removal Using Polarized Filtering in Document Scanning. U.S. Patent No. 8,174,739.* U.S. Patent and Trademark Office, Washington, D.C., May 8, 2012; http://assignments.uspto.gov/assignments/q?db=pat&pat=7561312
13. Roccetti, M., Marfia, G., and Zanichelli, M. The art and craft of making the tortellino: Playing with a digital gesture recognizer for preparing pasta culinary recipes. *Computers In Entertainment 8*, 4 (Dec. 2010), 28.
14. Rosner, D.K. and Ryokai, K. Spyn: Augmenting knitting to support storytelling and reflection. In *Proceedings of the 10th International Conference on Ubiquitous Computing* (Seoul, Sept. 21–24). ACM Press, New York, 2008, 340–349.
15. Schatzki, T.R. A primer on practices: Theory and research. Chapter in *Practice-Based Education: Perspective and Strategies*, J. Higgs, R. Barnett, S. Billett, and M. Hutchings, Eds. Sense Publishers, Rotterdam, 2012, 13–26.

Daniela K. Rosner (dkrosner@uw.edu) is an assistant professor of engineering and co-director of the Tactile and Tactical Design Lab at the University of Washington, Seattle.

Marco Roccetti (roccetti@cs.unibo.it) is a full professor in the Computer Science Department of the University of Bologna, Bologna, Italy.

Gustavo Marfia (gustavo.marfia@unibo.it) is an assistant professor in the Department for Life Quality Studies at the University of Bologna, Bologna, Italy.

DOI:10.1145/2500874

Atomically consistent memory services provide resiliency in dynamic settings.

BY PETER MUSIAL, NICOLAS NICOLAOU, AND ALEXANDER A. SHVARTSMAN

Implementing Distributed Shared Memory for Dynamic Networks

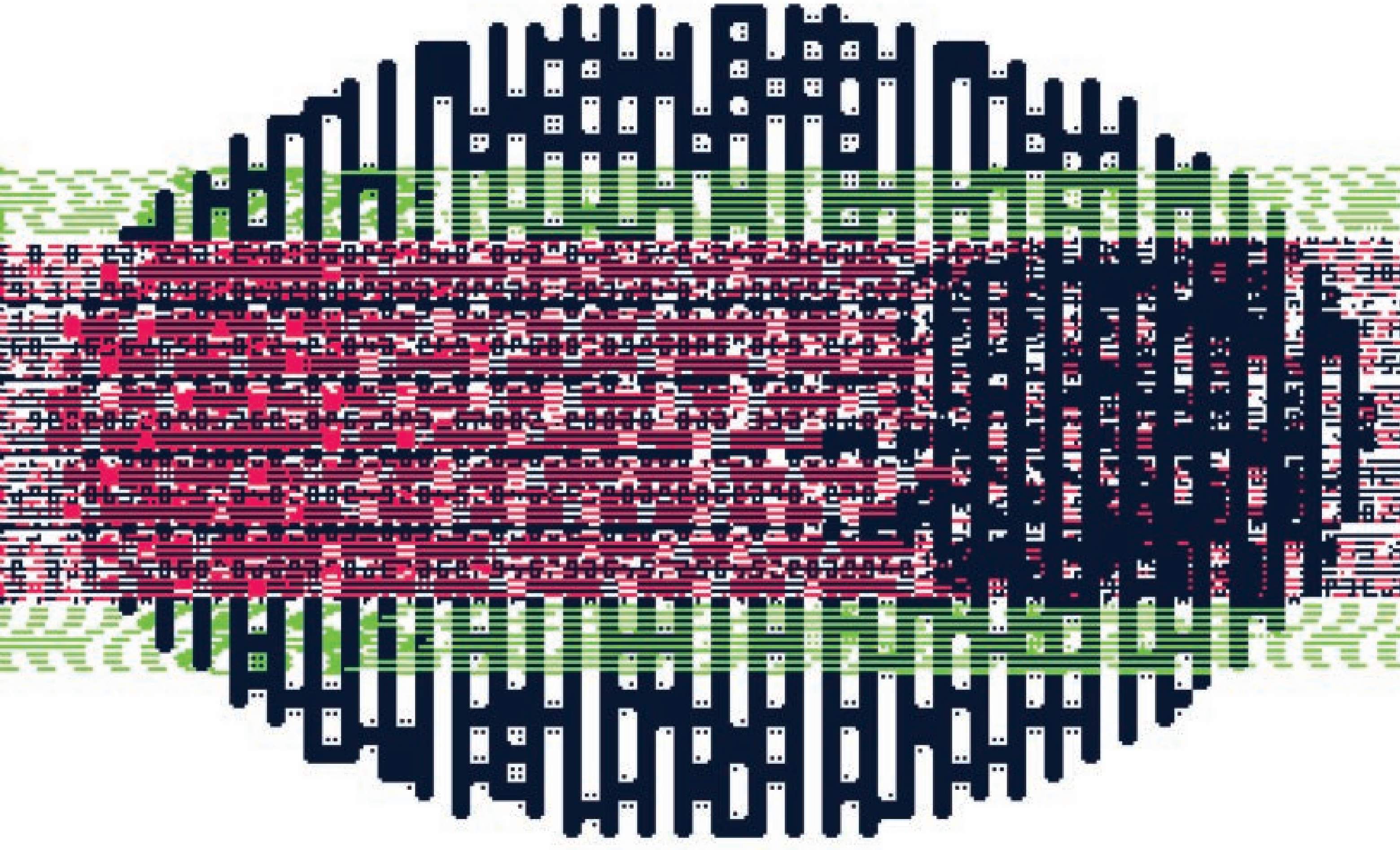
READING, 'RITING, AND 'RITHMETIC, the three Rs underlying much of human intellectual activity, not surprisingly, also stand as a venerable foundation of modern computing technology. Indeed, both the Turing machine and von Neumann machine models operate by reading, writing, and computing, and all practical uniprocessor implementations are based on performing activities structured in terms of the three Rs. With the advance of networking technology, communication became an additional major systemic activity. However, at a high level of abstraction, it is apparently still more natural to think in terms of reading, writing, and computing.

While it is difficult to imagine distributed systems—such as those implementing the World Wide Web—without communication, we often imagine browser-based applications that operate by retrieving (that is, reading) data, performing computation, and storing (that is, writing) the results. In this article, we deal with the storage of shared readable and writable data in distributed systems with the focus on implementations that provide resiliency and consistency in dynamic settings, namely, in systems that are subject to perturbations in the underlying distributed platforms composed of computers and networks that interconnect them. The perturbations include failures of individual computers, dynamically changing collections of computers participating in the system, and failures and delays in the communication medium.

Shared storage services are at the core of most information-age systems. Shared memory systems surveyed here provide objects supporting two access operations: *read* that obtains the current value of the object, and *write* that replaces the old value of the object with a new value. Such objects are often called *registers*. Although we do

» key insights

- **Dynamic shared memory (DSM) systems support objects that are accessed using read and write operations while guaranteeing consistency and availability despite the perturbations in the underlying distributed platform.**
- **Developers can use DSM to implement distributed systems using the shared memory paradigm and focus their work on the system features without undue concerns about the low-level message passing setting, asynchrony, or failures.**
- **DSM systems incorporate transparent runtime reconfiguration of the collections of object replicas and admit a variety of replica implementations ranging from being based on nodes in a storage area network to mobile devices in an ad hoc network.**



IMAGES BY IGNOTUS THE MAGE

not include more complicated object semantics, such as transactions or integrated read-modify-write operations, there are common implementation challenges that any distributed storage system needs to resolve. Imagine a storage system that is implemented as a central server. The server accepts client requests to perform operations on its data objects and returns responses. While this is conceptually simple, this approach already presents two major problems. The first is the central server is a performance bottleneck. The second is the server is a single point of failure. The quality of service in such an implementation degrades rapidly as the number of clients grows, and the service becomes unavailable if the server crashes (imagine how inadequate a Web news service would be were it implemented as a central server). Thus the system must, first of all, be available. This means it must provide its services despite failures within the scope of its specification, for example, the system must be able to mask certain server and communication failures. The system must also support multiple concurrent accesses without imposing unreasonable degradation in performance. The only way to guarantee availability is through *redundancy*, that is, to use multiple servers and to replicate the contents of the objects among these servers. Moreover, the replication must be done at geographically distributed and distinct network locations, where the disconnection or failures of certain subsets of data servers can be masked by the system.

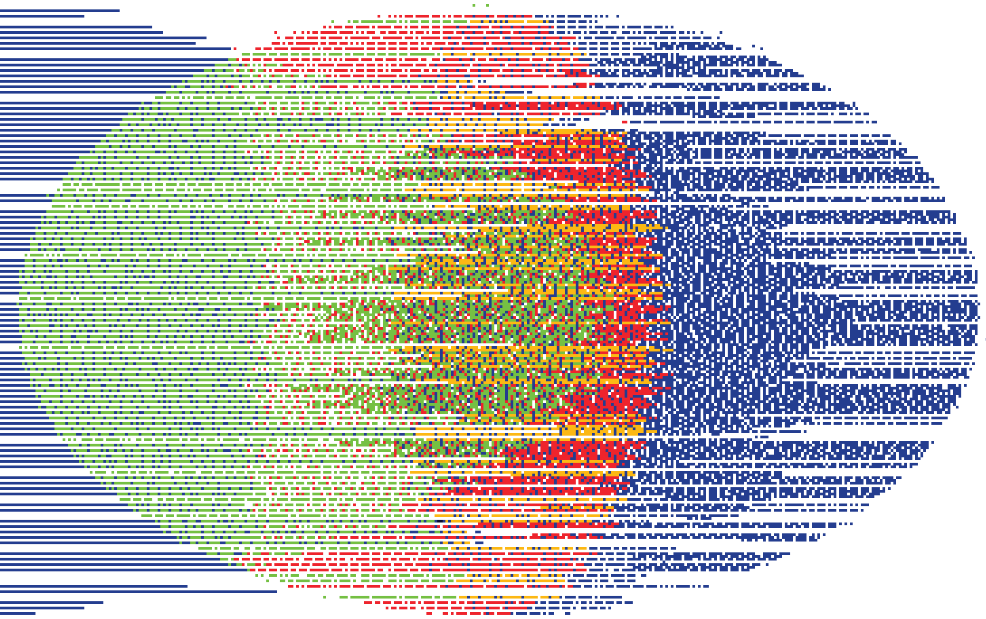
It is also critically important to ensure data longevity. A storage system may be able to tolerate failures of some servers, but over a long period it is conceivable that all servers may need to be replaced, because no servers are infallible, and due to planned upgrades. Additionally, in mobile settings, for example, search-and-rescue or military operations, it may be necessary to provide migration of data from one collection of servers to another, so the data can move as the needs dictate. Whether our concern is data longevity or mobility, the storage system must provide seamless runtime migration of data: one cannot stop the world and reconfigure the system in response to failures and changing environment.

A major problem that comes with replication is consistency. How does the system find the latest value of a replicated object? This problem was not present with a central server implementation: the server always contains the latest value. In a replicated implementation, one may attempt to consult all replicas in search of the latest value, but this approach is expensive and not fault-tolerant as it assumes that all replicas are accessible. In any case, none of the implementation issues should be a concern for the clients of the distributed memory service. What the clients should expect to see is the illusion of a single-copy object that serializes all accesses so that each read operation returns the value of the preceding write operation, and that this value is at least as recent as that returned by any preceding read. More generally, the behavior of an object, as observed externally, must be consistent with the abstract sequential data type of the object, and in developing applications that use such objects the clients must be able to rely on the abstract data type of the object. This notion of consistency is formalized as atomicity[28] or, equivalently, as *linearizability*.[25]

While there is no argument that atomicity is the most convenient notion of consistency, we note that weaker notions have also been proposed and implemented, motivated primarily by efficiency considerations. For example, several less-than-intuitive consistency definitions emerged from the domain of multiprocessor memory systems, precipitating an anecdotal opinion that "no one can figure out these consistency models, but the memory access is fast." Atomicity provides strong guarantees, making it more expensive to provide than weaker consistency guarantees.[4] Weak consistency also needs to be considered in the partitionable network setting. Eric Brewer conjectured[7] that distributed storage systems cannot simultaneously provide *consistency*, *availability*, and *partition-tolerance*; known as the "CAP conjecture," this was later formalized as a theorem.[23] Thus weaker consistency models need to be considered in some cases.[8] We take the view that it is nevertheless important to provide simple and intuitive atomic consistency.

The situation reminiscent of the early days of programming languages when it was argued that assembly language is better because one can generate more efficient code, or the early days of graphical user interfaces when it was argued that command-line interfaces are better because they consume fewer resources. It is conceivable that such arguments will also pass in the case of atomicity. In the setting where network partitions are a concern, two approaches are considered. When partitions are intermittent and short-lived, one can deal with them as long(er) network delays, delegating the repair issues to lower level communication services. Otherwise, when partitions can be permanent, strong consistency can still be guaranteed using the approaches that constrain the service to primary partitions, with subsequent integration when partitions are merged. Relevantly, in a recent keynote address,[30] ACM's 2008 A.M. Turing Award recipient Barbara Liskov remarked that atomicity is not cheap but if we do not guarantee it, this creates headaches for developers.

Interestingly, some existing storage systems provide data access primitives implementing atomic *read-modify-write* operations. Such access primitives are much stronger than separate *read* and *write* primitives we consider here. Implementing this type of consistency is expensive, and at its core requires atomic updates that in practice are implemented by either reducing parts of the system to a *single-writer* model (for example, Microsoft's Azure[9]), depending on clock synchronization hardware (Google's Spanner[13]), or relying on complex mechanisms for resolving event ordering such as vector clocks (Amazon's Dynamo[15]). Our exposition of atomic read/write storage illustrates challenges that are common to all distributed storage systems.



Distribution and Consistency

Here, we describe a general distributed setting for implementing consistent shared memory services.

Modeling distributed platforms. We model the system as a collection of interconnected computers, or nodes, that communicate by sending point-to-point messages. Each node has a unique identifier (such as IP address), local storage, and it can perform local computation. A node may fail by crashing at any point of the computation. Any node that crashes stops operating: it does not perform any local computation, it does not send any messages, and any messages sent to it are not delivered.

The system is asynchronous, and the nodes have no access to a global clock or synchronization mechanisms. This means that relative processing speeds at the nodes can be arbitrary, and that the nodes do not know the upper bound on time that it takes to perform a local computation. The message delays can also be arbitrary, and the nodes do not know bounds on message latency (although such bounds may exist). Thus the algorithms may not make assumptions about global time or delays, since the rate of processing at each node and message latencies are unknown.

We assume that messages can be reordered in transit, however, the messages cannot be corrupted, duplicated, or generated spontaneously. If a message is received then it must have been previously sent. The messages are not lost, but message loss can be modeled as long delays (we do not address techniques for constructing more dependable communication services, for example, by using retransmission or gossip).

We categorize a distributed networked system as either static or dynamic as follows. In the static networked system the set of participating nodes is fixed, and each node may know the identity of every other participant; crashes (or voluntary departures) may remove nodes from the system. In the dynamic networked system the set of nodes may be unbounded, and the set of participating nodes may completely change over time as the result of crashes, departures, and new nodes joining; such changes may occur at any time.

At this point we do not state assumptions on the magnitude of failures or "churn" in a dynamic setting, and we postpone this discussion until specific solutions are presented. Regardless of the dynamics, we require that consistency of the data is preserved. Perturbations in the computing medium may negatively affect the performance of the memory services we consider, however memory access operations are guaranteed to terminate under certain assumptions. For example, a static memory service guarantees that operations terminate when a majority of replicas is active and when the network delays are bounded. Dynamic memory services relax the static majority assumption and instead assume that dynamically changing subsets of replicas (for example, dynamic quorums) are active during certain periods. If these assumptions are not satisfied, such services still guarantee atomicity, however some operations may be slow or may not terminate.

Overall, we consider the participating nodes to be "good citizens" in that they cooperate willingly when they are able to, and they do not act maliciously by intentionally or accidentally performing incorrect computation. Thus we do not present techniques for dealing with malicious behaviors. Instead we direct the interested reader to related work that considers participants that may act nefariously, for example, Rodrigues et al.,[35] and Martin and Alvisi.[34]

Distributed shared memory and consistency. A distributed shared memory service emulates a shared memory space comprised of readable and writable objects (often called registers) over a networked platform consisting of distributed network nodes that communicate by message passing. Service implementations use replication to ensure survivability and availability of the objects, but the service makes this invisible to the clients. The content of each object is replicated across several servers or replica hosts. Clients invoke read and write operations on the objects, where the clients that perform read operations are called readers, and those that perform write operations are called writers (a client may be both a reader and a writer).

A major problem that comes with replication is consistency. How does the system find the latest value of a replicated object?

In response to client requests the service invokes a protocol that involves communication with the replica hosts. This protocol determines the consistency guarantees of the memory system. Atomic consistency definition involves "shrinking" the duration of each operation in any execution to a chosen serialization point between the operation's invocation and response, and requiring that the ordering of the operations according to the serialization points preserves their real-time ordering, and the resulting behavior of the object is consistent with its sequential specification. In particular, if a read is invoked after a write completes, then the read is guaranteed to return either the value of that write, or a value written by subsequent write that precedes the read. Additionally, if a read is invoked after another read completes, it returns the same or a "newer" value than the preceding read. (We present this more formally in an appendix available in the ACM Digital Library.) It is due to these natural properties that atomicity is the most convenient and intuitive consistency guarantee.

Atomic registers were introduced by Lamport[28] (who also defined weaker notions, such as safety and regularity). Herlihy and Wing[26] proposed an equivalent definition called linearizability, that also extends atomicity to arbitrary data types. Given that atomicity is the strongest notion of consistency providing the most useful access semantics, we focus on the atomic shared memory systems.

Building Block: Shared Memory in Static Networked Systems

Algorithms designed for static settings must accommodate some dynamic behaviors, such as asynchrony, transient failures, and permanent crashes within certain limits. Here, we present a shared memory service for such settings for two reasons: to set the stage for more dynamic services in the next section, and to focus on the approach to atomicity that is also used in several dynamic services. The first comprehensive survey of implementations of shared memory in the static setting is presented by Chockler et al.[12] Some replica access protocols given in that work can be used as building blocks for dynamic implementations, however, algorithms designed for the static setting cannot be used directly due to their inability to handle dynamically changing sets of replicas. One such algorithm is the seminal implementation of atomic shared memory given by Attiya, Bar-Noy, and Dolev,[3] and commonly referred to as the ABD algorithm; this work won the Dijkstra Prize in 2011.

The algorithm implements atomic memory, where replication helps achieve fault-tolerance and availability. The system tolerates f replica node crashes, provided the number of replicas n is such that $n > 2f$. Our presentation includes extensions to the original algorithm following Lynch and Shvartsman.[32]

Each replica host i maintains the local value of the register, $value_i$ and a tag associated with the replica, $tag_i = \langle seq, pid \rangle$, where seq is a sequence number and pid is the identifier of the node that performed the write of that value. Tags are compared lexicographically. Each new write assigns a unique tag, where the originating node id is used to break ties. These tags in effect determine an ordering of the write operations, and therefore determine the values that subsequent read operations return.

Read and write operations are similar, each is implemented in terms of two phases, a `Get` phase that queries replicas for information, and a `Put` phase that propagates information to replicas. Each phase protocol ensures some *majority* participates in the communication exchange: first the messages are sent to all replica hosts, then the replies are collected until some majority of replicas responds. Recall that since $n > 2f$, a majority of non-crashed replicas always exists. Thus, each phase terminates after a single communication round, and any operation terminates after two communication rounds.

The correctness of this implementation, that is, atomicity, follows from the fact that for any pair of operations when one follows another, at least one correct replica is involved in the `Put` phase of the first operation and in the `Get` phase of the second; this ensures the second operation will always "see" the value that is at least as recent as that of the most recent preceding operation. We point out the only property used in conjunction with the majorities is that any two majorities intersect. If waiting for a majority is too tasking, one can use quorum systems instead.[32,38] (Details and the complete pseudo-code are in the appendix available in the ACM Digital Library.)

The clients of a shared memory service should expect to see the illusion of single-copy objects that serialize all accesses.

Emulating Shared Memory in Dynamic Networked Systems

Here, we present several approaches for providing consistent shared memory in more dynamic systems, that is, where nodes may not only crash or depart voluntarily, but where new nodes may join the service. In general, the set of object replicas can substantially evolve over time, ultimately migrating to a completely different set of replica hosts. The ABD algorithm cannot be used directly in such settings because it relies on the majority of original replica hosts to always be available. In order to use an ABD-like approach in dynamic settings, one must provide some means for managing the collections of replica hosts, and to ensure that readers and writers contact suitable such collections.

It is noteworthy that dealing with dynamic settings and managing collections of nodes does not directly address the provision of consistency in memory services. Instead, these issues are representative of the broader challenges present in the realm of dynamic distributed computing. It is illustrative that implementations of consistent shared memory services can sometimes be constructed using distributed building blocks, such as those designed for managing collections of participating nodes, for providing suitable communication primitives, and for reaching agreement (consensus) in dynamic distributed settings. A tutorial covering several of these topics is given by Aguilera et al.[1]

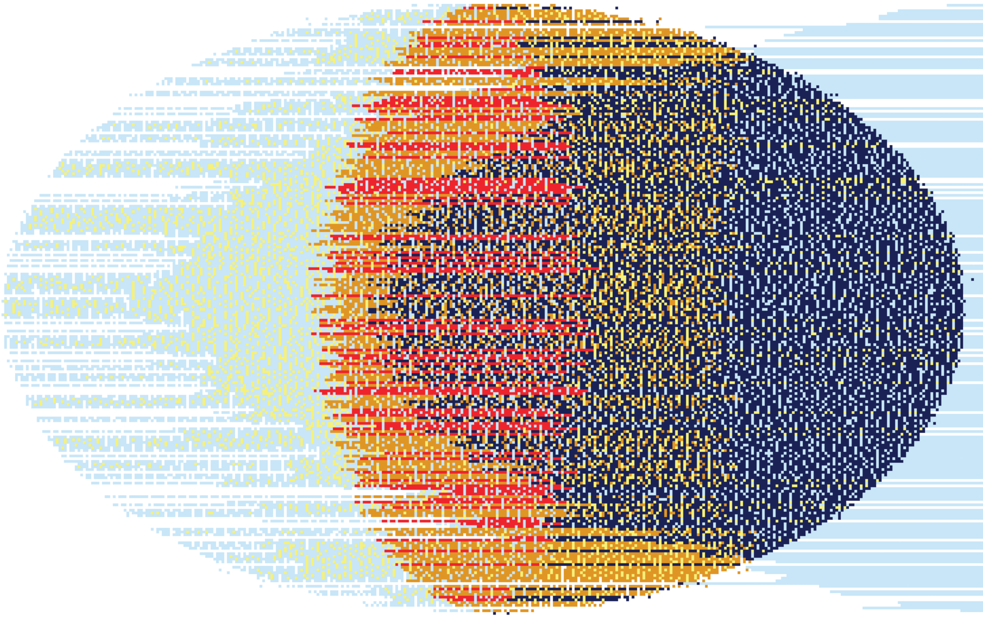
We start by presenting the consensus problem because it provides a natural basis for implementing an atomic memory service by establishing an agreed-upon order of operations, and because consensus is used in other ways in atomic memory implementations. Next, we present group communication services (GCS) solutions that use strong communication primitives, such as totally ordered broadcast, to order operations. Finally, we focus on approaches that extend the ideas of the ABD algorithm to dynamic settings with explicit management of the evolving collections of replica hosts.

Consensus. Reaching agreement in distributed settings is a fundamental problem of computer science. The agreement problem in distributed settings is called consensus. Here, a set of processes needs to agree on a value, with nodes proposing several values for consideration. Any solution must have the following properties: *Agreement*: no two processes decide on different values; *Validity*: the value decided was proposed by some process; *Termination*: all correct processes reach a decision. Consensus is a powerful tool in designing distributed services,[33] however, consensus is a notoriously difficult problem in asynchronous systems where termination cannot be guaranteed in the presence of even a single process crash;[19] thus consensus must be used with care. (One approach to solving consensus is based on introducing "failure detectors" that provide, possibly limited, information about the nodes in the system.[10])

Consensus algorithms can be used directly to implement an atomic data service by letting the participants use consensus to agree on a global total ordering of all operations.[29] The correctness (atomicity) in this approach is guaranteed regardless of the choice of a specific consensus implementation, but the understanding of the underlying platform characteristics can guide the choice of the implementation for the benefit of system performance (for a tour de force of implementations, see Lynch[33]). Nevertheless, using consensus for each operation is a heavy-handed approach, especially given that perturbations may delay or even prevent termination. Thus, when using consensus, one must avoid invoking it in conjunction with individual memory operations, and make operations independent of the termination of consensus.

We note that achieving consensus is a more difficult problem than implementing atomic read/write objects, in particular, consensus cannot be solved for two or more processes by using atomic read/write registers.[31]

Group communication services. Among the most important building blocks for distributed systems are group communication services



(GCSs)[5] that enable processes located at different nodes of a network to operate collectively as a group. The processes do this by using a GCS multicast service to send messages to all members of the group. GCSs offer various guarantees about the order and reliability of message delivery.

The basis of a GCS is a *group membership service*. Each process, at each time, has a unique view of the group that includes a list of the processes that are members of the group. Views can change over time, and may become different at different processes. Another important concept introduced by the GCS approach is virtual synchrony, where an essential requirement is that processes that proceed together through two consecutive views deliver the same set of messages between these views. This allows the recipients to take coordinated action based on the message, the membership set, and the rules prescribed by the application.[5]

GCSs provide one approach for implementing shared memory in dynamic networks. This can be done, for example, by implementing a global totally ordered multicast service on top of a view-synchronous GCS[18] (where there is a total order on the messages associated with each view, and each participant receives a prefix of this order). The ordered multicast is used to impose an order on the memory access operations, yielding atomic memory. The main disadvantage in such solutions is that in most GCS implementations, forming a new view takes a substantial amount of time, and client memory operations are delayed (or aborted) during the view-formation period. New view formation normally involves several rounds of communication, and in typical GCS implementations performance is degraded even if only one node crashes. In memory services it is desirable to ensure that read and write operations make progress during reconfiguration, and it is important to tolerate a modest number of failures without imposing a performance penalty.

Another approach is to integrate a GCS with the ABD algorithm. For example, the dynamic primary configuration GCS as describe by De Prisco et al.[14] implements atomic memory by using techniques of Attiya[3] within each configuration. Here, a configuration combines a group view with a quorum system. Configurations can change either due to the dynamic nature of the system or because a different quorum system is desired, and like other solutions based on GCSs, reads and writes are delayed during reconfiguration. Lastly, any new configuration must satisfy certain intersection properties with respect to the previous configurations. This impacts the flexibility of reconfiguration and requires intermediate configuration changes in reaching the desired configuration.

A general methodology for dynamic service replication is presented in Birman.[6] This reconfiguration model unifies the virtual synchrony approach with state machine replication, as used in consensus solutions, in particular, Paxos.[29]

DynaStore algorithm. DynaStore[2] is an implementation of a dynamic atomic memory service for multi-writer/multi-reader objects. It integrates the ABD algorithm and allows reconfiguration of the collection of replica hosts without the use of consensus.

The participants start with a default local configuration, that is, some common set of replica hosts. The algorithm supports three kinds of operations: *read*, *write*, and *reconfig*. The read and write operations involve two phases, and in the absence of reconfigurations, the protocol is similar to ABD: it uses majorities of replicas, where each replica maintains the value of the object and the associated tag.

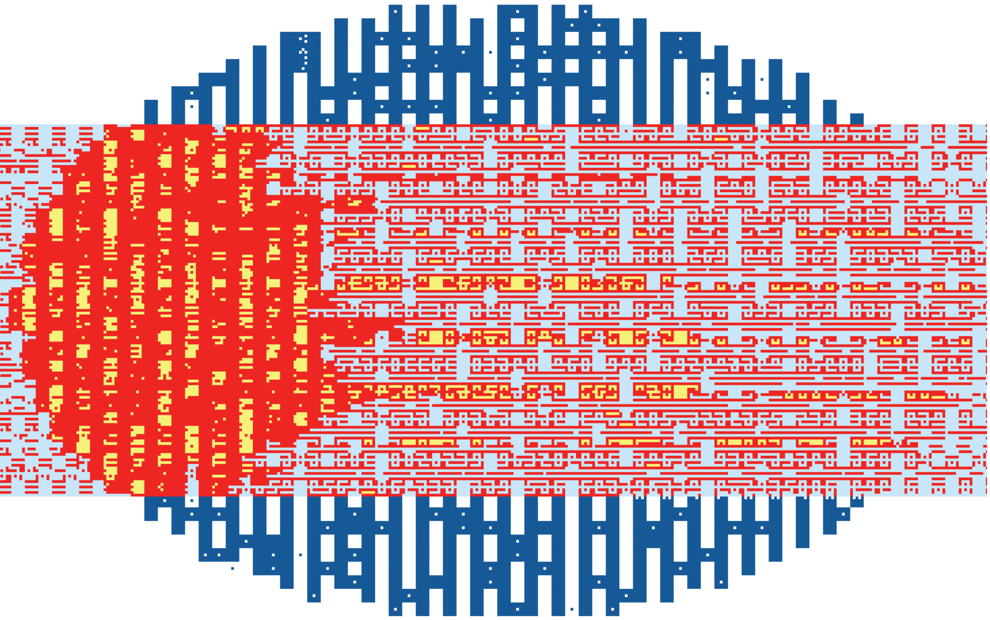
If a participant wishes to change its current configuration, it uses the reconfig operation and supplies with it a set of incremental changes with elements of the form (+, *i*), indicating the addition of replica host *i*, and (–, *j*), indicating the removal of host *j*. The implementation of *reconfig* consists of two phases (described below) and uses a distributed weak snapshot service to announce the locally originating changes by means of the update primitive, and obtain the changes submitted by other members of the configuration by means of the scan primitive. The snapshot itself is not atomic as it does not globally order updates, and because scan is not guaranteed to reflect all previously completed updates. Yet, the snapshot service is sufficient for establishing a certain *directed acyclic graph* (DAG) that is stored locally as part of the state of each participant. Vertices of the graph correspond to configurations that can be produced by means of changes that in turn correspond to the edges.

The implementation of *reconfig* involves traversals of such DAG's, representing possible sequences of changed configurations. In each traversal the DAG may be revised to remultiple changes, submitted at different hosts, to the same configuration. The assumption that a majority of the involved hosts are not removed and do not crash ensures there is a path through the DAG that is guaranteed to be common among all hosts. Interestingly, the hosts themselves do not learn of this common path, however, traversing all paths ensures the common path is also traversed. The traversal terminates when a sink node is reached. An additional assumption that there is a finite number of reconfigurations ensures termination.

Now we return to the two-phase structure of *reconfig*. The goal of the first phase is similar to the `Get` phase of ABD: discover the latest value-tag pair for the object. The goal of the second phase is similar to the `Put` phase of ABD: convey the latest value-tag pair to a suitable majority of replica hosts. The main difference is that these two phases are performed in the context of applying the incremental changes to the configuration, while at the same time discovering the changes submitted by other participants. This essentially "bootstraps" possible new configurations. Given that all of this is done by traversing all possible paths—and thus configurations—in the DAG's ensures the common path is also traversed.

Finally, we provide additional details for the read and write operations. The *read* follows the implementation of *reconfig*, with the differences being: (a) the set of configuration changes is empty, and (b) the discovered value is returned to the client. The *write* also follows the implementation of *reconfig*, with the differences being: (a) the set of changes is empty, (b) a new, higher tag is produced upon the completion of the first phase, and (c) the new value-tag pair is propagated in the second phase. Note that while the set of configuration changes is empty for *read* and *write*, both operations may discover changes submitted elsewhere, and help with the bootstrapping of revised configurations. As with *reconfig*, stable majorities ensure nothing blocks the protocols from progressing, and the finite number of reconfigurations guarantees termination.

It is worth reiterating that Dyna Store implementation does not incorporate an agreement protocol for reconfiguration. On the other hand, reconfigurations are accomplished by additions and removals of individual nodes and this may lead to larger overheads as compared to approaches that evolve the system by replacing a complete configuration with another. Thus the latency of *read* and *write* operations are more dependent on the rate of reconfigurations (we touch on this later when discussing DynaDisk). Finally, in order to guarantee termination, DynaStore assumes reconfigurations eventually subside.



The Rambo framework. Rambo is a dynamic memory service supporting multi-reader/multi-writer objects;[24] Rambo stands for Reconfigurable Atomic Memory for Basic Objects. This algorithm uses configurations, each consisting of a set of replica hosts plus a quorum system defined over these hosts, and supports reconfiguration, by which configurations can be replaced. Notably, any quorum configuration may be installed at any time, and quorums from distinct configurations are not required to have non-empty intersections. The algorithm ensures atomicity in all executions.

During quiescent periods when there are no reconfigurations, the algorithm operates similarly to the ABD algorithm[3] (generalized as in Lynch and Shvartsman[32]): each of the two phases involves interaction with one complete quorum in the current configuration. New participants join the service by means of message handshakes with at least one existing participant (this involves no reconfiguration).

Any participant may crash at any time. To enable long-term operation of the service, quorum configurations can be reconfigured. Reconfigurations are performed concurrently with any ongoing read and write operations, and do not directly affect such operations. Additionally, multiple reconfigurations may be in progress concurrently. Reconfiguration involves two decoupled protocols: introduction of a new configuration by the component called *Recon*, and upgrade to the new configuration and garbage collection of obsolete configuration(s).

We now discuss this in more detail. The main data structure maintained by each participant is the configuration map, or *cmap*, that stores a sequence of configurations, where for node *i*, $cmap_i$ (k) is the configuration number *k*.

This sequence evolves as new configurations are introduced by Recon and as old configurations are garbage collected. It is possible for participants to have differing views on what is in each *cmap*, however Recon always emits a unique new configuration *k* to be stored in every $cmap_i$ (k). This is done as follows. Any node *i* that is a member of its latest known configuration $c = cmap_i (k-1)$ can propose a new configuration at any time. Different proposals are reconciled by executing consensus among the members of *c* (here, consensus can be implemented, for example, using a version of Paxos[29]). Although a consensus execution may be slow—in fact, in some situations, it may not even terminate—this dors not delay read and write operations (provided at least one quorum set is intact for the involved configurations during the operations). Note the members of the new configuration may or may not know the latest value of the object. It is the duty of a decoupled *upgrade* protocol to garbage collect old configurations and propagate the information about the object to the latest locally known configuration. Here, a two-phase algorithm first tells a quorum of each older configuration about the new one and obtains the latest object information, then propagates this information to a quorum of the new configuration and removes the obsolete configurations.

GeoQuorums is an approach to implementing atomic shared memory on top of a physical platform that is based on mobile nodes moving in arbitrary patterns.

It is possible for the participants to have multiple active (non-garbage-collected) configurations in their *cmap* if reconfigurations occur quickly, or if configuration upgrade lags behind. In this case the two-phase protocol that implements read and write operations behaves as follows. The first phase gathers information from the quorums of active configurations; the second phase propagates information to the quorums of active configurations. Note that during each phase new configurations may be discovered. To handle this, each phase is terminated by a fixed-point condition that involves a quorum from each active configuration.

Revisiting reconfiguration in Rambo, memory access operations are decoupled from reconfigurations enabling operations to terminate even when Recon may be slow due to its use of consensus. The reconfiguration itself involves two separate activities: a consistent sequence of configurations is issued with the help of consensus, and then the upgrade process finalizes the reconfiguration. In some settings it may be advantageous to integrate these two activities for specialized performance reasons, for example, as in RDS service.[11]

Lastly, Rambo can be viewed as a framework for refinements and optimizations, some of which were implemented in a prototype system, for example, Georgiou et al.[20] Next, we describe a remolding of the Rambo framework for ad hoc mobile networks.

GeoQuorums[16] is an approach to implementing atomic shared memory on top of a physical platform that is based on mobile nodes moving in arbitrary patterns. An ad hoc network uses no preexisting infrastructure; instead, the mobile nodes that cooperate to route communication from sources to destinations form the network. GeoQuorums can be viewed as a system of two layers, where the top layer implements a dynamic replicated storage system, and the bottom layer provides object replicas in terms of stationary focal points that are implemented by the mobile nodes.

The focal points are geographic areas of interest that are normally "populated" by mobile nodes. This may be a road junction, a scenic observation point, or a water resource in the desert. Mobile nodes in the vicinity of a focal point participate in implementing a stationary virtual object, called the focal point object. The implementation at each focal point supports a local broadcast service, *LBcast*, that provides reliable, totally ordered broadcast. LBcast is used to implement a type of replicated state machine, one that tolerates joins and leaves of mobile nodes. If every mobile node leaves the focal point, the focal point object fails.

Next, this approach defines a collection of quorum systems over the focal points. Each quorum system involves two sets, called get-quorums and put-quorums, with the property that every get-quorum intersects every put-quorum. The use of quorums enables the service to tolerate the failure of a limited number of focal point objects. For reasons of performance, or in response to periodic migration of mobile nodes, different quorum systems can be installed.

To facilitate communication with the focal point objects, GeoQuorums assumes the availability of a GeoCast service (as in Imielinski[26]) that enables message delivery to the geographic area of a focal point. With this setting, one can use the Rambo framework as the top layer to implement an atomic memory system with focal points serving as replica hosts. Motivated by simplicity and efficiency considerations, the GeoQuorums approach makes additional modifications. The first deals with reconfiguration, and the second affects how read and write operations are processed.

Dynamic atomic memory services provide data consistency and availability despite the perturbations of the underlying distributed platforms.

GeoQuorums introduced the first general reconfiguration capability that does not rely on consensus. The algorithm reconfigures among a finite number of predetermined configurations, and instead of consensus it uses a two-phase protocol that is similar to the upgrade protocol in Rambo. Here in the first phase, the invoker contacts any complete get-quorum and put-quorum of all preceding configurations (note that at most one pair of messages per focal point is needed even if the finite number of possible configurations is large), then in the second phase information is conveyed to any complete put-quorum of the next configuration.

GeoQuorums implements a modified approach to read and write operations that allows some operations to complete in just one phase. This is accomplished for the case of writes with the help of a global positioning system (GPS) clock to generate tags for the written values, thus ordering writes. This obviates the need for the phase that in other implementations determines the highest tag, and the write protocol here performs just a single *put* phase that interacts with any put-quorum in the current configuration. If the write detects a concurrent reconfiguration, it also awaits response from put-quorums in every configuration (this involves at most one contact with each focal point). Once the write completes, the tag becomes confirmed. For the case of reads, this protocol involves one or two phases. The first, *get*, phase proceeds as a typical query phase to obtain the value with the maximum tag from some complete get-quorum; if a concurrent reconfiguration is detected, then the phase also awaits responses from one get-quorum from each configuration (again, at most one message exchange per focal point). Once the *get* phase completes, and it is determined the maximum obtained tag is *confirmed*, the read terminates. Otherwise, the read performs the *put* phase that propagates the maximum tag-value pair to some put-quorum. The information about the *confirmed* tags is propagated through the system to enable single-phase read operations.

The use of separate get-quorums and put-quorums allows one to tune the system performance depending on the balance between reads and writes. When there are more writes than reads, the system can reconfigure to use larger get-quorums and smaller put-quorums because writes access only the put-quorums. When reads are more prevalent, the system can use smaller get-quorums and larger put-quorums because some reads access just the get-quorums.

GeoQuorums uses real-time timestamps (provided through GPS) to expedite read and write operations, allowing the writes and some reads to complete in a single round. Lastly, the assumption that there is a fixed set of focal points limits the system's evolvability, but allows the algorithm to reconfigure without the use of consensus.

From Theory to Practice

The precise consistency guarantees, the tolerance to failures, and the ability to operate in dynamic environments prompted researchers to construct exploratory implementations of some services. We have already mentioned the exploratory implementation of Rambo variants. Here, we present two additional implementations. The first is a distributed disk array implementation[36] that derives algorithmic ideas from Rambo. The second implementation[37] is based on the DynaStore algorithm.

Federated Array of Bricks (FAB)[36] is a storage system developed and evaluated at HP Labs. FAB deals with disk storage, where the basic objects are *logical blocks*. The goal of the implementation is to outperform traditional master-slave replication by distributing the workload and handling failures and recoveries without disturbing the client requests.

FAB uses computers, called *bricks*, equipped with commodity disks and a network interface. To achieve distribution, FAB splits the storage into logical blocks, and using an erasure-coding algorithm replicates each logical block to a subset of bricks. The system is based on majority quorum systems, where to ensure longevity of the service a reconfiguration algorithm moves data when bricks are added or removed. A client performs read and write operations by sending a request to a brick that specifies the logical block it wants to obtain or modify. The brick then determines the set of bricks that store this block and runs the read or write protocol involving this set of bricks as specified by the Rambo algorithm, also using tag-value pairs to determine the order of the written values. Writes require two phases to complete whereas reads may take one or two phases. One-phase reads are possible when the reader receives the same tag from all bricks. To improve efficiency the reader requests the actual block contents only from an idle, active replica, and requests only tags from other members of a quorum.

Evaluations of the implementation showed that FAB performance is similar to the centralized solutions, while offering at the same time continuous service and high availability.

DynaDisk[37] is an evaluation implementation of the DynaStore algorithm.[2] The implementation was tested in a local area network and adopts the data-centric approach, where the replica hosts are passive network storage devices.

The implementation was designed to reconfigure the service either asynchronously without the use of consensus, or partially synchronously with the help of consensus.

The evaluation showed that in the absence of reconfigurations the two versions of the algorithm have similar read and write operation latencies. When multiple reconfigurations occur concurrently, it was observed that the asynchronous, consensus-free approach has a significant negative effect on the latency of read and write operations that are concurrent with reconfigurations. The explanation here is that the consensus-free algorithm must examine multiple possible configurations, whereas an algorithm that incorporates reconfiguration with consensus normally deals with a single configuration on which all clients agree. Reconfiguration latency on the other hand is somewhat better in the consensus-free version of DynaDisk when many reconfigurations occur simultaneously. In such situations, the consensus-based DynaDisk may take longer to reach a decision.

Discussion

We presented several approaches to implementing atomically consistent memory services in dynamic distributed systems. In such systems the collections of participating nodes may evolve over time due to failures, voluntary or planned departures of nodes, and new nodes join the computation. We focused on atomic consistency because it is an intuitive notion that, despite replication, provides equivalence with serial object specifications. Such services make it easier to construct distributed applications, especially given the perception that using the shared memory paradigm is easier than using the message passing paradigm in designing distributed algorithms.

The approaches presented in this article are representative of the different design choices available for implementing distributed memory services. Each of these has its pros and cons. For instance, solutions based on consensus, while conceptually simple, typically use coordinators during certain stages, and the performance may critically depend on the availability of the coordinator. Substantial delays may be incurred during coordinator failover, even if a large majority of hosts do not fail. Group communication services is an effective building block for low-latency networks with high link availability, however they suffer from high overheads during view changes that occur even if the number of failures is relatively small, but spread over longer stretches of time. Theoretically elegant approaches that use incremental host replacement such as DynaStore, quorum replacement such as Rambo, and focal points such as GeoQuorums, may be more complex to implement, but have greater flexibility and allow system-specific optimizations. Achieving good performance also depends on stability of the deployment platform, assumptions on failure rates, and other factors, such as those considered in a tailored implementation of the FAB system.

Regardless of how the reconfiguration of the set of replica hosts is done, any practical implementation must address the challenge of deciding *when* to reconfigure. Here, one may delegate the decision to individual hosts, for example, a node joining the service causes reconfiguration, or a node implicitly causes reconfiguration when its failure is detected. Although such approaches may be simple and effective when perturbations are infrequent and the set of replicas is small, they may cause undue overheads when nodes continue joining and leaving the service even though some core set of hosts is stable and sufficient for providing good service. Another approach is to leave the decision for when to reconfigure to another distributed service that monitors the performance of the memory system and decides when to reconfigure based on these observations and on speculative forecasts. This is a more complicated solution, but it has the potential of providing superior quality of service. Additional consideration is selecting a suitable set of hosts. Just because a node is interested in serving as a replica host does not mean its wish must be granted. Here, the external service that decides when to reconfigure can also decide on the target set of nodes. Note that no agreement on the target set is required—all memory services are able to deal with the situations when several target sets are proposed.

The dynamic atomic shared memory services guarantee consistency in all executions, regardless of the magnitude or frequency of replica host failures. Termination of read and write operations, however, is conditioned on restricting failures. For static systems, generally this restriction can be easily formulated: here, any minority subset of hosts is allowed to fail. For dynamic systems the constraints on failure patterns are much more involved and depend on the specific algorithmic approaches; the reader is referred to the cited articles for additional details.

With the advent of Cloud services, distributed storage services are bound to continue attract attention. The technical challenges and performance overheads may be the reasons why the existing distributed storage solutions shy away from atomic consistency guarantees. Commercial solutions, such as Google's File System (GFS),[22] Amazon's Dynamo,[15] and Facebook's Cassandra,[28] provide less-than-intuitive, unproved guarantees. The concepts discussed in this survey are echoed in the design decisions of production systems. For instance, consensus is used in GFS[22] to ensure agreement on system configuration as it is done in Rambo; global time is used in Spanner[13] as it is done in GeoQuorums; replica access protocols in Dynamo[15] use quorums as in some approaches surveyed here. These examples provide motivation for pursuing rigorous algorithmic approaches in the study of consistent data services for dynamic networked systems.

Consistent storage systems continues to be an area of active research and advanced development, and there are good reasons to believe that as high-performance dynamic memory systems with superior fault-tolerance become available, they will play a significant role in the construction of sophisticated distributed applications. The demand for implementations providing atomic read/write memory will ultimately be driven by the needs of distributed applications that require provable consistency and performance guarantees.

Acknowledgments

This work was supported in part by the NSF award 1017232. We thank Jennifer Welch and the anonymous reviewers for numerous insightful comments.

References

1. Aguilera, M., Keidar, I., Martin, J.-P. and Shraer, A. Reconfiguring replicated atomic storage: A tutorial. *Bulletin of the EATCS 102* (Oct. 2010), 84–108.
2. Aguilera, M.K., Keidar, I., Malkhi, D. and Shraer, A. Dynamic atomic storage without consensus. *JACM 58* (Apr. 2011), 7:1–7:32.
3. Attiya, H., Bar-Noy, A. and Dolev, D. Sharing memory robustly in message-passing systems. *JACM 42*, 1 (Jan. 1995), 124–142.
4. Attiya, H. and Welch, J.L. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst. 12*, 2 (May 1994), 91–122.
5. Birman, K. A history of the virtual synchrony replication model. Replication: Theory and Practice, *LNCS* vol. 5959 (2010), 91–120.
6. Birman, K., Malkhi, D. and Renesse, R.V. Virtually synchronous methodology for dynamic service replication. Technical report, MSR-TR-2010-151, Microsoft Research, 2010.
7. Brewer, E.A. Towards robust distributed systems, July 2000.
8. Brewer, E.A. Pushing the cap: Strategies for consistency and availability. *IEEE Computer 45*, 2 (2012), 23–29.
9. Calder, B. et al. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of SOSP '11* (Oct 23-26, 2011), 143–157.
10. Chandra, T.D., Hadzilacos, V. and Toueg, S. The weakest failure detector for solving consensus. *JACM* (1996), 685–722.
11. Chockler, G., Gilbert, S., Gramoli, V., Musial, P.M. and Shvartsman, A.A. Reconfigurable distributed storage for dynamic networks. *J. Parallel and Distributed Computing 69*, 1 (2009), 100–116.
12. Chockler, G., Guerraoui, R., Keidar, I. and Vukolić, M. Reliable distributed storage. *IEEE Computer*, 2008.
13. Corbett, J.C. et al. Spanner: Google's globally distributed database. In *Proceedings of the 10th USENIX Symp. On Operating Sys. Design and Implementation* (2012), 251–264.
14. De Prisco, R., Fekete, A., Lynch, N.A. and Shvartsman, A.A. A dynamic primary configuration group communication service. In *Proceedings of the 13th Int-l Symposium on Distributed Computing*. Springer-Verlag, 1999, 64–78.
15. DeCandia, G. et al. Dynamo: Amazon's highly available key-value store. In *Proceedings of SIGOPS Oper. Syst. Rev. 41*, 6 (Oct. 2007), 205–220.
16. Dolev, S., Gilbert, S., Lynch, N., Shvartsman, A. and Welch, J. GeoQuorums: Implementing atomic memory in ad hoc networks. In *Proceedings of the 17th International Symposium on Distributed Computing* (2003), 306–320.
17. Dutta, P., Guerraoui, R., Levy, R.R. and Vukolić, M. Fast access to distributed atomic memory. *SIAM J. Comput. 39*, 8 (Dec. 2010), 3752–3783.
18. Fekete, A., Lynch, N. and Shvartsman, A. Specifying and using a partitionable group communication service. *ACM Trans. Comput. Syst. 19*, 2 (2001), 171–216.
19. Fischer, M.J., Lynch, N.A. and Paterson, M.S. Impossibility of distributed consensus with one faulty process. *JACM 32*, 2 (1985), 374-382.
20. Georgiou, C., Musial, P.M. and Shvartsman, A.A. Developing a consistent domain-oriented distributed object service. *IEEE Transactions of Parallel and Distributed Systems 20*, 11 (2009), 1567–1585.
21. Georgiou, C., Musial, P.M. and Shvartsman, A.A. Fault-tolerant semifast implementations of atomic read/write registers. *J. Parallel and Distributed Computing 69*, 1 (Jan. 2009), 62–79.
22. Ghemawat, S., Gobioff, H. and Leung, S.-T. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (2003), 29–43.
23. Gilbert, S. and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News 33* (June 2002), 51–59.
24. Gilbert, S., Lynch, N. and Shvartsman, A. RAMBO: A robust, reconfigurable atomic memory service for dynamic networks. *Distributed Computing 23*, 4, (Dec. 2010), 225–272.
25. Herlihy, M.P. and Wing, J.M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Programming Languages and Systems 12*, 3 (July 1990), 463–492.
26. Imieliński, T. and Navas, J.C. GPS-based geographic addressing, routing, and resource discovery. *Commun. ACM 42*, 4 (Apr. 1999), 86–92.
27. Lakshman, A. and Malik, P. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev. 44*, 2 (Apr. 2010), 35–40.
28. Lamport, L. On interprocess communication. Part I: Basic formalism. *Distributed Computing 2*, 1 (1986), 77–85.
29. Lamport, L. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (1998), 133–169.
30. Liskov, B. The power of abstraction. In Proceedings of the 24th Int-l Symposium Distributed Computing. N.A. Lynch and A.A. Shvartsman, Eds. *LNCS*, vol. 6343, Springer, 2010.
31. Loui, M.C. and Abu-Amara, H.H. Memory requirements for agreement among unreliable asynchronous processes. In *Parallel and Distributed Computing, Vol 4 of Advances in Computing Research*. F.P. Preparata, Ed. JAI Press, Greenwich, Conn., 1987, 163–183.
32. Lynch, N. and Shvartsman, A. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Symposium on Fault-Tolerant Computing*. IEEE, 1997, 272–281.
33. Lynch, N.A. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.
34. Martin, J.-P. and Alvisi, L. A framework for dynamic byzantine storage. In *Proc. Intl. Conf. on Dependable Systems and Networks*, 2004.
35. Rodrigues, R., Liskov, B., Chen, K., Liskov, M. and Schultz, D. Automatic reconfiguration for large-scale reliable storage systems. *IEEE Trans. on Dependable and Secure Computing 9*, 2 (2012), 145–158.
36. Saito, Y., Frølund, S., Veitch, A., Merchant, A. and Spence, S. Fab: Building distributed enterprise disk arrays from commodity components. *SIGARCH Comput. Archit. News 32*, 5 (Oct. 2004), 48–58.
37. Shraer, A. Martin, J.-P., Malkhi, D. and Keidar, I. Data-centric reconfiguration with network-attached disks. In *Proceedings of the 4th Int'l Workshop on Large Scale Distributed Systems and Middleware* (2010), ACM, 22–26.
38. Vukolić, M. Quorum systems: With applications to storage and consensus. *Synthesis Lectures on Distributed Computing Theory 3*, (Jan. 3, 2012). 1–146.

Peter Musial (pmmusial@csail.mit.edu) is a principal software engineer at EMC and a CSAIL affiliate at MIT, Cambridge, MA.

Nicolas Nicolaou (nicolasn@cs.ucy.ac.cy) is a visiting lecturer in the Computer Science Department of University of Cyprus, Nicosia, Cyprus.

Alexander A. Shvartsman (aas@cse.uconn.edu) is a professor of computer science and engineering at the University of Connecticut, Storrs, CT.

research highlights

DOI:10.1145/2602759

To view the accompanying paper, visit doi.acm.org/10.1145/2602758 rh

Technical Perspective
Motion Fields for Interactive Character Animation

By Michiel van de Panne

COMPUTER GRAPHICS HAS long sought to create realistic depictions of humans, both in appearance and in their movement. A good starting point for animation has been to begin by collecting data from the real world about how people move. The use of prerecorded motion clips has been a standard building block for producing the motion of virtual characters for use in film, games, and interactive simulations. Of course, it is impossible to capture every motion a person might perform, and so animation methods attempt to do as much as possible with the limited movements that one might be able to record using either a motion capture setup or handmade animation.

With motion clips in hand, the basic idea for creating new motions is simple: play sequences of the prerecorded clips while being careful to only switch to a new clip when this can be done without causing any noticeable artifacts. Smoothly blending between motions during the transition period can further help to increase the number of feasible transitions between clips. For interactive settings such as games and simulations, control can be added by making decisions about which clip to play next based on the user's goals, such as a particular desired walking direction or speed. A set of motion clips and their feasible clip transitions naturally form a "motion graph," which is a directed graph of motions with demarcated transition points where motions can join or split off. Modern game engines exploit various extensions of these ideas, such as the further use of blending to interpolate between motions, for example, walks of different speeds or turn rates, and the use of independent motion clips for the upper body and lower body when this can be done without introducing artifacts. However, the ongoing motion is still constructed from an underlying set of motions that are following their preset paths.

The preceding model comes with some obvious limitations, however. First, if the user commands a sudden turn or change in speed, the motion must wait until the next available clip transition before anything can be done. At this point, many game engines choose to rapidly cut or blend to the desired motion rather than producing a higher quality continuous motion that feels sluggish to the player because of the longer response time. Second, the motion can never really leave the confines of the prerecorded paths, at least not without further manual crafting of motion clip blends.

The following paper is exciting because it proposes to largely discard the idea of motion clips, and instead it effectively treats the motion clip as a set of independent motion vectors, where a motion vector consists of a character pose and its related velocities. In their ensemble, these high-dimensional motion vectors define a motion field that governs how the state of a character evolves over time. For any given state of the character, the passive dynamics is defined by the mean motion field, which can be reconstructed as a weighted average of the set of motion vectors that are nearest the current state. However, the mean motion does not allow for any control, and so produces realistic but unresponsive motion. One of the key insights of this paper, then, is to also use the neighboring motion vectors to define a discrete set of actions that each represent an alternative choice for guiding the future evolution of the motion. With this ability to "steer" the motion of the character in hand, reinforcement learning can be used to great effect to precompute the optimal actions for given task goals, such as walking in a given direction at a given speed.

The following paper is exciting because it treats the motion clip as a set of independent motion vectors, where a motion vector consists of a character pose and its related velocities.

The beauty of the method is that the synthesized motions are free to depart from the prerecorded motion clips, and yet they can still be guided to satisfy the goals of the motion. Part of the magic comes from the available control actions being implicitly deconstructed from the motion field and therefore not needing to be explicitly specified. Also exciting is that the method supports the use of largely unstructured motion data and is conceptually easy to integrate with other existing kinematic and dynamic methods. The paper convincingly demonstrates the benefits of rethinking existing representations for problems in order to arrive at new solutions. Lastly, it demonstrates a compelling example of how reinforcement learning methods can be applied convincingly in high-dimensional settings. C

Michiel van de Panne (van@cs.ubc.ca) is a professor and associate head for Research and Faculty Affairs in the CS department at University of British Columbia, Vancouver, Canada.

DOI:10.1145/2602758

Motion Fields for Interactive Character Locomotion

By Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović

Abstract

We propose a novel representation of motion data and control of virtual characters that gives highly agile responses to user input and allows a natural handling of arbitrary external disturbances. In contrast to traditional approaches based on replaying segments of motion data directly, our representation organizes samples of motion data into a high-dimensional generalization of a vector field that we call a *motion field*. Our runtime motion synthesis mechanism freely flows through the motion field in response to user commands. The motions we create appear natural, are highly responsive to real-time user input, and are not explicitly specified in the data.

1. INTRODUCTION

Whenever a video game contains a character that walks or runs, it requires some method for interactively synthesizing this locomotion. This synthesis is more involved than it might at first appear, since it requires both the creation of visually accurate results and the ability to interactively *control* which motions are generated. The standard techniques for achieving this create realistic animation by directly (or nearly directly) replaying prerecorded clips of animation. They provide control by carefully specifying when it is possible to transition from playing one clip of animation to another. The synthesized motions are thus restricted to closely match the prerecorded animations.

True human motion, however, is a highly varied and continuous phenomenon: it quickly adapts to different tasks, responds to external disturbances, and in general is capable of continuing locomotion from almost any initial state. As video games increasingly demand that characters move and behave in realistic ways, it is important to bring these properties of natural human motion into the virtual world. Unfortunately, this is easier said than done. For instance, despite many advances in character animation techniques, creating highly agile and realistic interactive locomotion controllers remains a common but difficult task.

We propose a new motion representation for interactive character animation, termed a *motion field* that provides two key abilities: the ability for a user to control the character in real time and the ability to operate in the fully continuous configuration space of the character—thereby admitting *any* possible pose for a character as a valid state to animate from, rather than being restricted to poses that are near those in the prerecorded motion data. Although there exist techniques that allow one or the other of these abilities, it is the combination of the two that allows for highly agile controllers that can respond to user commands in a short amount of time.

More specifically, a motion field is a mapping that associates each possible configuration of a character with a *set* of motions describing how the character is able to move from this current state. In order to generate an animation we select a single motion from this set, follow it for a single frame, and repeat from the character's resulting state. The motion of the character thus "flows" through the state space according to the integration process, similar to a particle flowing through a force field. However, instead of a single fixed flow, a motion field allows multiple possible motions at each frame. By using reinforcement learning to intelligently choose between these possibilities at runtime, the direction of the flow can be altered, allowing the character to respond optimally to user commands.

Because motion fields allow a range of actions at every frame, a character can immediately respond to new user commands rather than waiting for predetermined transition points between different animation clips as in motion graphs. This allows motion field-based controllers to be significantly more agile than their graph-based counterparts. By further altering this flow with other external methods, such as inverse kinematics or physical simulation, we also can directly integrate these techniques into the motion synthesis and control process. Furthermore, since our approach requires very little structure in the motion capture data that it uses, minimal effort is needed to generate a new controller.

2. RELATED WORK

In the past ten years, the primary approaches for animating interactive characters have been based on carefully replaying precaptured clips of animation. By detecting where transitions can be made from one clip of animation to another without requiring a visually obvious "jump" in the animation, these methods allow both realistic motion and real-time control. In order to better understand our motion fields approach described later, it is useful to have a geometric picture of how these types of methods operate. The full space of all the possible ways in which a character's body can be posed contains dozens of dimensions, but one can picture it abstractly as a two-dimensional (2D) plane. Each point in this plane then describes a single static pose for the character, while an animation is described by a continuous path. If we imagine this 2D plane as something like a university quad, then the animation could be "traced

The original version of this paper was published in the *Proceedings of the ACM SIGGRAPH Asia 2010*.

out" by following the path made while walking through this quad.

For approaches that rely on directly replaying clips of precaptured animation,[1, 6, 7] one can imagine these recorded clips as forming a number of paved paths through the quad. Each path represents the animation data, and the points where paths meet or branch correspond to points where the motion can smoothly transition from one animation to another. Intuitively, these clip-based approaches correspond to the restriction that one always walks on the paved paths, thus representing the ways motions can be synthesized as a graph. Although this restriction means that any synthesized motions closely match the data, most of the space of possible motions is rendered off-limits. This has the disadvantage of making it difficult to quickly respond to changes in user commands or unexpected disturbances, since a change to the motion can only happen when a new edge is reached.[11, 16] Second, because the motions are restricted to the clips that constitute the graph, it is difficult to couple these methods to physical simulators and other techniques which perturb the state away from states representable by the graph. More generally, it is very hard to use a graph-based controller when the character starts from an arbitrary state configuration.[20] In essence, our approach of motion fields addresses these problems by allowing the animation to veer "off the paths," and it uses a set of precaptured animations only as a rough guide in determining what motion to synthesize.

Although a number of methods have been proposed to alleviate some of the representational weaknesses of pure graph-based controllers, including parameterized motion graphs,[5, 14] increasing the numbers of possible transitions,[2, 19] and splicing rag doll dynamics in the graph structure,[20] the fundamental issue remains: unless the representation prescribes motion at every continuous state in a way that is controllable in real time, the movement of characters will remain restricted. Hence, even when the method anticipates some user inputs,[11] the character may react too slowly, or transition too abruptly because there is no shorter path in the graph. Similarly, when methods anticipate some types of upper-body pushes,[2] the character may not react at all to hand pulls or lower-body pushes.

Another group of methods uses nonparametric models to learn the dynamics of character motion in a fully continuous space.[3, 17, 18] These techniques are generally able to synthesize starting from any initial state, and lend themselves well to applying physical disturbances[18] and estimating a character's pose from incomplete data.[3] These models are used to estimate a single "most likely" motion for the character to take at each possible state. This precludes the ability to optimally control the character. The primary difference between our work and these is that instead of building a model of the most probable single motion, we attempt to model the set of possible motions at each character state, and only select the single motion to use at runtime by using principles from optimal control theory. This allows us to interactively control the character while enjoying the benefits of a fully continuous state space. Our work combines the concepts of near-optimal character control present in graph-based methods with those of nonparametric motion estimation techniques.

Although our controllers are kinematic (in that they do not directly incorporate physics except via the data on which they are built), dynamic controllers have been extensively explored as an alternative method of character animation. In principle, such controllers offer the best possibility for highly realistic interactive character animation. However, high-fidelity physically based character animation is harder to attain because physics alone does not tell us about the muscle forces needed to propel the characters, and the design of agile, lifelike, fully-dynamic characters remains an open challenge.

3. MOTION FIELDS

Interactive applications such as video games require characters that can react quickly to user commands and unexpected disturbances, all while maintaining believability in the generated animation. An ideal approach would fully model the complete space of natural human motion, describing every conceivable way that a character can move from a given state. Rather than confine motion to canned motion clips and transitions, such a model would enable much greater flexibility and agility of motion through the continuous space of motion.

Although it is infeasible to completely model the entire space of natural character motion, we can use motion capture data as a local approximation. We propose a structure called a *motion field* that finds and uses motion capture data similar to the character's current motion at any point. By consulting similar motions to determine which future behaviors are plausible, we ensure that our synthesized animation remains natural: similar, but rarely identical to the motion capture data. This frees the character from simply replaying the motion data, allowing it to move freely through the general vicinity of the data. Furthermore, because there are always multiple motion data to consult, the character constantly has a variety of ways to make quick changes in motion.

In this section we will describe how we define a motion field from a set of example animation clips. For the moment we will ignore the question of how a motion field can be used to interactively control a character, instead focusing on synthesizing noninteraction animations. We achieve this by organizing a set of example animation clips into a motion field to describe a single "flow"—allowing an animation to be generated starting from any possible character pose. In Section 4 we will then describe how this technique can be extended to allow for the character to be interactively controlled in real time.

3.1. Preliminary definitions

Motion states. We represent the states in which a character might be configured by the pose and the velocity of all of a character's joints. A *pose* $x = (x_{\text{root}}, p_0, p_1, \ldots, p_n)$ consists of a 3D root position vector x_{root}, a root orientation quaternion p_0, and joint rotation quaternions $p_1, \ldots, p_n$. The root point is located at the pelvis. A *velocity* $v = (v_{\text{root}}, q_0, q_1, \ldots, q_n)$ consists of a 3D root displacement vector v_{root}, root displacement quaternion q_0, and joint displacement quaternions $q_1, \ldots, q_n$—all found via finite differences. Given two poses x and x', we can compute this finite difference as

$$v = x' \ominus x = (x'_{\text{root}} - x_{\text{root}}, p'_0 p_0^{-1}, p'_1 p_1^{-1}, \ldots, p'_n p_n^{-1})$$

By inverting the above difference, we can add a velocity v to a pose x to get a new displaced pose $x' = x \oplus v$. We can also interpolate multiple poses or velocities together $\left(\sum_{i=1}^{k} w_i x_i\right.$ or $\left.\sum_{i=1}^{k} w_i v_i\right)$ using linear interpolation of vectors and unit quaternion interpolation[13] on the respective components of a pose or velocity. We use $\ominus$, $\oplus$, and $\sum$ in analogy to vector addition and subtraction in Cartesian spaces, but with circles to remind the reader that we are working mostly with quaternions.

Finally, we define a *motion state* $m = (x, v)$ as a pose and an associated velocity, computed from a pair of successive poses x and x' with $m = (x, v) = (x, x' \ominus x)$. The set of all possible motion states forms a high-dimensional continuous space, where every point represents the state of our character at a single instant in time. A path or trajectory through this space represents a continuous motion of our character. When discussing dynamic systems, this space is usually called the phase space. However, because our motion synthesis is kinematic, we use the phrase *motion space* instead to avoid confusion.

Motion database. Our approach takes as input a set of motion capture data and constructs a set of motion states $\{m_i\}_{i=1}^{n}$ termed a *motion database*. Each state m_i in this database is constructed from a pair of successive frames x_i and x_{i+1} by the aforementioned method of $m_i = (x_i, v_i) = (x_i, x_{i+1} \ominus x_i)$. We also compute and store the velocity of the *next* pair of frames, computed by $y_i = x_{i+2} \ominus x_{i+1}$. Generally, motion states, poses, and velocities from the database will be subscripted (e.g., m_i, x_i, v_i, and y_i), while arbitrary states, poses, and velocities appear without subscripts.

Similarity and neighborhoods. Central to our definition of a motion field is the notion of the *similarity* between motion states. Given a motion state m, we compute a *neighborhood* $\mathcal{N}(m) = \{m_i\}_{i=1}^{n}$ of the k most similar motion states via a k-nearest neighbor query over the database.[12] In our tests we use $k = 15$. We calculate the (dis-)similarity by

$$d(m, m') = \sqrt{\begin{array}{r} \beta_{\text{root}} \left\| v_{\text{root}} - v'_{\text{root}} \right\|^2 \ + \\ \beta_0 \left\| q_0(\hat{u}) - q'_0(\hat{u}) \right\|^2 \ + \\ \sum_{i=1}^{n} \beta_i \left\| p_i(\hat{u}) - p'_i(\hat{u}) \right\|^2 \ + \\ \sum_{i=1}^{n} \beta_i \left\| (q_i p_i)(\hat{u}) - (q'_i p'_i)(\hat{u}) \right\|^2 \ + \end{array}} \qquad (1)$$

where $\hat{u}$ is some arbitrary unit length vector; $p(\hat{u})$ means the rotation of $\hat{u}$ by p; and the weights $\beta_{\text{root}}, \beta_0, \beta_1, \ldots, \beta_n$ are tunable scalar parameters. In our experiments, we set β_i as bone lengths of the body at the joint i in meters, and β_{root} and β_0 are set to 0.5. Intuitively, setting β_i to the length of its associated bone de-emphasizes the impact of small bones such as the fingers. Note that we factor out root world position and root yaw orientation (but not their respective velocities).

Similarity weights. Since we allow the character to deviate from motion states in the database, we frequently have to interpolate data from our neighborhood $\mathcal{N}(m)$. We call the weights $[w_0, \ldots, w_k]$ used for such interpolation *similarity weights* since they measure similarity to the current state m:

$$w_i = \frac{1}{\eta} \frac{1}{d(m, m_i)^2} \qquad (2)$$

where m_i is the ith neighbor of m and $\eta = \sum_i \frac{1}{d(m, m_i)^2}$ is a normalization factor to ensure the weights sum to 1.

3.2. Motion synthesis

Actions. The value of a *motion field* $\mathcal{A}$ at a motion state m is a set of control actions $\mathcal{A}(m)$, determining which states the character can transition to in a single frame's time. Each of these actions $a \in \mathcal{A}(m)$ specifies a convex combination of neighbors $a = [a_1, \ldots, a_k]$ (with $\sum a_i = 1$ and $a_i > 0$). By increasing the values of some of the a_i weights over others, the direction of an action is biased to better match the directions of the associated neighbors (Figure 1). Given one particular action $a \in \mathcal{A}(m)$, we then determine the next state m' using a transition or *integration function* $m' = (x', v') = \mathcal{I}(x, v, a) = \mathcal{I}(m, a)$. Letting i range over the neighborhood $\mathcal{N}(m)$, we use the function

$$\mathcal{I}(m, a) = \left(x \oplus \sum a_i v_i, \sum a_i y_i\right) \qquad (3)$$

Unfortunately, this function frequently causes our character's state to drift off into regions where we have little data about how the character should move, leading to unrealistic motions. To correct for this problem, we use a small drift correction term that constantly tugs our character toward the closest known motion state $\overline{m} = (\overline{x}, \overline{v})$ in the database. The strength of this tug is controlled by a parameter $\delta = 0.1$:

$$\mathcal{I}(m, a) = (x \oplus v', y') \qquad (4)$$

$$v' = (1 - \delta)\left(\sum_{i=1}^{k} a_i v_i\right) \oplus \delta\left((\overline{x} \oplus \overline{v}) \ominus x\right) \qquad (5)$$

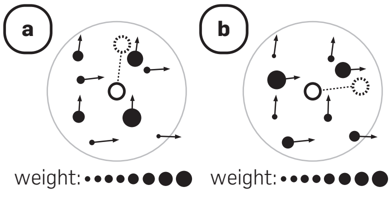
$$y' = (1 - \delta)\left(\sum_{i=1}^{k} a_i y_i\right) \oplus \delta \overline{y} \qquad (6)$$

Passive action selection. Given a choice of action we now know how to generate motion, but which action should we pick? This question is primarily the subject of Section 4. However, we can quickly implement a simple solution using the similarity weights (Equation (2)) as our choice of action. This choice results in the character meandering through the data, generating streams of realistic (albeit undirected) human motion.

4. CONTROL

As described in Section 3, at each possible state of the character, a motion field gives a *set* of actions that the character

Figure 1. *Control using action weights*. By reweighting the neighbors (black dots) of our current state (white dot), we can control motion synthesis to direct our character toward different next states (dashed dots). (a) Weights resulting in an upward motion. (b) Weights resulting in a motion to the right.



can choose from in order to determine their motion over the next frame. In general, which particular action from this set should be chosen depends on the user's current commands. Deciding on which action to choose in each state in response to a user's commands is thus key in enabling real-time interactive locomotion controllers.

4.1. Markov decision processes control

From a user's perspective, the easiest way to control a virtual character is through high-level commands such as "turn to the right" or "walk backwards." Ultimately, these high-level commands must be boiled down to the low-level actions needed to execute them. In the case of motion fields, these low-level actions are chosen at each frame. Because these low-level choices occur over such a short time scale, it is not always directly obvious which choices should be made in order to satisfy a user's high-level command. In order to allow for easy user control, it is therefore necessary to connect the single-frame actions used in a motion field with their long-term consequences.

Efficiently planning for the long-term consequences of short-term actions is a standard problem in artificial intelligence. In particular, the domain of artificial intelligence known as *reinforcement learning* provides tools that can be naturally applied to controlling a character animated with a motion field. In order to apply these tools from reinforcement learning, we formulate the motion field control problem as a Markov decision process (MDP).

An MDP is a mathematical structure formalizing the concept of making decisions in light of both their immediate and long-term results. An MDP consists of four parts: (1) a state space, (2) actions to perform at each state, (3) a means of determining the state transition produced by an action, and (4) rewards for occupying desired states and performing desired actions. By expressing character animation tasks in this framework, we make our characters aware of long-term consequences of their actions. This is useful even in graph-based controllers, but vital for motion field controllers because we are acting every frame rather than every clip. For further background on MDP-based control, see Sutton and Barto,[15] or Treuille[16] and Lo and Zwicker[10] for their use in graph-based locomotion controllers.

States. Simply representing the state of a character as a motion state m is insufficient for interactive control, because we must also represent how well the character is achieving its user-specified task. We therefore add a vector of *task parameters* θ_T to keep track of how well the task is being performed, forming joint *task states* $s = (m, \theta_T)$. For instance, our direction-following task θ_T records a single number—the angular deviation from the desired heading. By altering this value, the user controls the character's direction.

Actions. At each task state $s = (m, \theta_T)$, a character in a motion field has a set of actions $\mathcal{A}(m)$ to choose from in order to determine how they will move over the next frame (Section 3). There are infinitely many different actions in $\mathcal{A}(m)$, but many of the techniques used to solve MDPs require a *finite* set of actions at each state. In order to satisfy this requirement for our MDP controller, we sample a finite set of actions $\mathcal{A}(s)$ from $\mathcal{A}(m)$. Given a motion state m, we generate k actions by modifying the similarity weights (Equation (2)). Each action is designed to prefer one neighbor over the others.

$$\left\{\frac{a_i}{\|a_i\|} \,\middle|\, a_i = (w_0,\ldots,w_{i-1},1,w_{i+1},\ldots,w_{k-1})\right\} \tag{7}$$

In other words, to derive action a_i simply set w_i to 1 and renormalize. This scheme samples actions that are not too different from the passive action at m so as to avoid jerkiness in the motion while giving the character enough flexibility to move toward nearby motion states.

Transitions. Additionally, we must extend the definition of the integration function $\mathcal{I}$ (Equation (4)) to address task parameters: $\mathcal{I}_s(s, a) = \mathcal{I}_s(m, \theta_T, a) = (\mathcal{I}(m, a), \theta'_T)$. How to update task parameters is normally obvious. For instance, in the direction-following task, where θ_T is the character's deviation from the desired direction, we simply adjust θ_T by the angle the character turned.

Rewards. In order to make our character perform the desired task we offer rewards. Formally, a *reward function* specifies a real number $R(s, a)$, quantifying the reward received for performing the action a at state s. For instance, in our direction-following task, we give a high reward $R(s, a)$ for maintaining a small deviation from the desired heading and a lower reward for large deviations. See Section 5 for the specific task parameters and reward functions we use in our demos.

4.2. Reinforcement learning

The goal of reinforcement learning is to find "the best" rule or *policy* for choosing which action to perform at any given state. A naïve approach to this problem would be to pick the action that yields the largest immediate reward—the *greedy policy*:

$$\pi_G(s) = \underset{a\in\mathcal{A}(s)}{\operatorname{argmax}}\, R(s,a) \tag{8}$$

Although simple, this policy is myopic, ignoring the future ramifications of each action choice. We already know that greedy graph-based controllers perform poorly.[16] Motion fields are even worse. Even for the simple task of changing direction, we need a much longer horizon than 1/30th of a second to anticipate and execute a turn.

Somehow, we need to consider the effect of the current action choice on the character's ability to accrue future rewards. A *lookahead policy* π_L does just this by considering the cumulative reward over future task states:

$$\pi_L(s) = \underset{a\in\mathcal{A}(m)}{\operatorname{argmax}}\left[R(s,a) + \max_{\{a_t\}}\sum_{t=1}^{\infty}\gamma^t R(s_t,a_t)\right] \tag{9}$$

with $s_1 = \mathcal{I}_s(s, a)$ and $s_t = \mathcal{I}_s(s_{t-1}, a_{t-1})$. γ is called the *discount factor* and controls how much the character focuses on short-term ($\gamma \to 0$) versus long-term ($\gamma \to 1$) reward.

As mentioned earlier, computing the lookahead policy involves solving for not only the optimal next action but also an *infinite* sequence of optimal future actions. Despite this apparent impracticality, a standard trick allows us to efficiently solve for the correct next action. The trick begins by defining a *value function* $V(s)$, a scalar-valued function representing the expected cumulative future reward received for acting optimally starting from task state s:

$$V(s) = \max_{a\in\mathcal{A}(m)} \sum_{t=0}^{\infty} \gamma R(s_t, a_t) \tag{10}$$

We will describe shortly how we represent and precompute the value function, but for the moment notice that we can now rewrite Equation (9) by replacing the infinite future search with a value function lookup:

$$\pi_L(s) = \operatorname*{argmax}_{a\in\mathcal{A}(m)} \left[R(s,a) + V\big(\mathcal{I}_s(s,a)\big)\right] \tag{11}$$

Now the lookahead policy is only marginally more expensive to compute than the greedy policy.

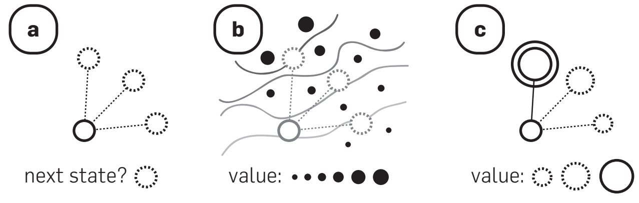
Value function representation and learning. Since there are infinitely many possible task states, we cannot represent the value function exactly. Instead, we approximate it by storing values at a finite number of task states s_i and interpolating to estimate the value at other points (Figure 2). We choose these task state samples by taking the Cartesian product of the database motion states m_i and a uniform grid sampling across the problem's task parameters (see Section 5 for details of the sampling). This sampling gives us high resolution near the motion database states, which is where the character generally stays. In order to calculate the value $V(s)$ of a task state not in the database, we interpolate over neighboring motion states using the similarity weights and over the task parameters multilinearly.

Given an MDP derived from a motion field and a task specification, we solve for an approximate value function in this form using *fitted value iteration*.[4] Fitted value iteration operates by first noting that Equation (11) can be used to write the definition of the value function in a recursive form. We express the value at a task state sample s_i recursively in terms of the value at other task state samples:

$$V(s_i) = R(s_i, \pi_L(s_i)) + V(\mathcal{I}_s(s_i, \pi_L(s_i))) \tag{12}$$

where $\pi_L(s_i)$ is as defined in Equation (11) and $V(\mathcal{I}_s(s_i, a))$ is computed via interpolation. We can solve for $V(s_i)$ at each sample state by iteratively applying Equations (11) and (12). We begin with an all-zero value function $V_0(s_i) = 0$ for each sample s_i. Then at each s_i, Equation (11) is used to compute $\pi_L(s)$ after which we use Equation (12) to determine an updated value at s_i. After all the s_i samples have been processed in this manner, we have an updated approximation of the value function. We repeat this process until convergence and use the last iteration as the final value function.

Figure 2. ***Action search using a value function.*** **(a) At every state, we have several possible actions (dashed lines) and their next states (dashed circles). (b) We interpolate the value function stored at database states (black points) to determine the value of each next state. (c) We select the highest value action to perform.**



Temporal value function compression. Unlike graph-based approaches, motion fields let characters be in constant transition between many sources of data. Consequently, we need access to the value function at all motion states, rather than only at transitions between clips. This fact leads to a large memory footprint relative to graphs. We offset this weakness with compression.

For the purpose of compression, we want to think of our value function as a collection of value functions, each defined over the space of task parameters. Without compression, we store one of these value subfunctions $V_{m_i}(\theta_T) = V(m_i, \theta_T)$ at every database motion state m_i (see Figure 3). Here, we observe that our motion states were originally obtained from continuous streams of motion data. At 30 Hz, temporally adjacent motion states and their value functions are frequently similar; we expect that V_{m_t} changes smoothly over "consecutive" motion states m_t relative to the original clip time. Exploiting this idea, we only store value functions at every Nth motion state, and interpolate the value functions for other database motion states (see Figure 4). We call these database states storing value functions "anchor" motion states. We compute the value function at the ith motion state between two anchors m_0 and m_N as

$$V_{m_i}(\theta_T) = \frac{N-i}{N} V_{m_0}(\theta_T) + \frac{i}{N} V_{m_N}(\theta_T) \tag{13}$$

We can learn a temporally compressed value function with a trivially modified form of the algorithm given in Section 4.2.1. Instead of iterating over all task states, we only iterate over those states associated with anchor motion states.

Figure 3. ***Uncompressed value function.*** **The value functions V_{m_i} are stored at every motion state m_i.**

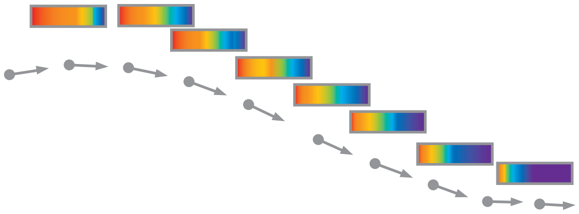
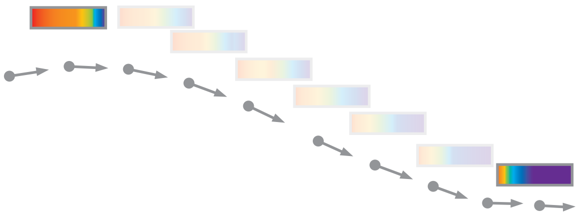


Figure 4. ***Value function with temporal compression.*** **The value functions at intermediate motion states are interpolated by the neighboring "anchor" motion states that have explicitly stored value functions.**



This technique allows the trade-off between the agility of a motion field-based controller and its memory requirements. Performing little or no temporal interpolation yields very agile controllers at the cost of additional memory, while controllers with significant temporal compression tend to be less agile. In our experiments, we found that motion field controllers with temporal compression are approximately as agile as graph-based controllers when restricted to use an equivalent amount of memory, and significantly more agile when using moderately more memory (see Section 5).

5. EXPERIMENTS

This section presents analysis on two important properties of motion fields—agility in responding to user directive changes and ability to respond to dynamic perturbation.

5.1. Agile responses to user control

Experiment setup. We created value functions for two example tasks: following an arbitrary user-specified direction and staying on a straight line while following the user direction (see Figure 5). The reward $R_{\text{direction}}$ for the direction task and the reward R_{line} for the line-following task are respectively defined as

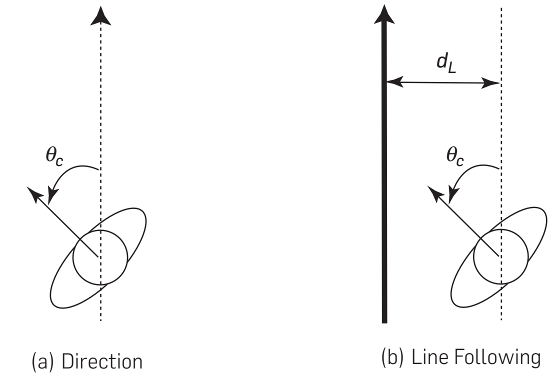
$$R_{\text{direction}}(m, \theta_c, a) = -|\theta_c| \qquad (14)$$

$$R_{\text{line}}(m, \theta_c, d_L, a) = -|\theta_c| - 0.05|d_L| \qquad (15)$$

Motion data setup. We used 142 seconds of motion data containing leisurely paced locomotion and quick responses to direction and line changes. We selected the source motion data with minimum care except to roughly cover the space of possible motion. The only manual preprocessing was foot contact annotation.

Value function computation. We use value iteration to calculate the value function. For the direction task, we store values for 18 uniformly sampled directions θ_c. For the line-following task, we take a Cartesian cross product sampling between 18 uniform θ_c samples and 13 uniform d_L samples spanning –2.0 to 2.0 meters. We set the discount factor to $\gamma = 0.99$. For each task, we also created "temporally compressed" versions of the value functions, where we set $N = 1, 10, 20, 30$ in Equation (13). Using value iteration to solve for the value function takes within 2 minutes if there is sufficient memory to cache the actions and transitions, and 3 hours otherwise. Distributing the value iteration updates over a cluster of computers can easily address these time and memory burdens.

Figure 5. ***Task parameters*. For the direction-following task (a), the difference in angle θ_c of the desired direction from the character facing direction is used. For the line-following task (b), distance to the desired line d_L is also considered with θ_c.**



(a) Direction

(b) Line Following

Response timing analysis.

Graph-based control vs motion field control. In order to compare how quickly the character can adjust to abruptly changing directives, we created a graph-based task controller[8] using the same motion data, tasks, and reward functions. In order to maximize agility, we allowed a wide range of up to ±45 degrees of directional warping on clips, and gave minimal importance to the physicality cost (see Lee et al.[8] for details). Figure 6 shows typical responses to changing user directions. For both tasks, the motion fields demonstrated much quicker convergence to new goals, as shown in the accompanying video and Table 1.

Effect of value function compression. We recorded response times using the compressed value functions on uniformly sampled user direction changes. With increasing degree of compression the system still reliably achieved user goals, but gradually lost agility in the initial response (see Table 1). We ran a similar experiment for the line-following task. We uniformly sampled user direction changes as well as line

Figure 6. ***Response time*. Direction adjustment over time with three consecutive direction changes within 4.23 seconds. The motion field control adjusts in a significantly shorter time period than the graph-based control.**

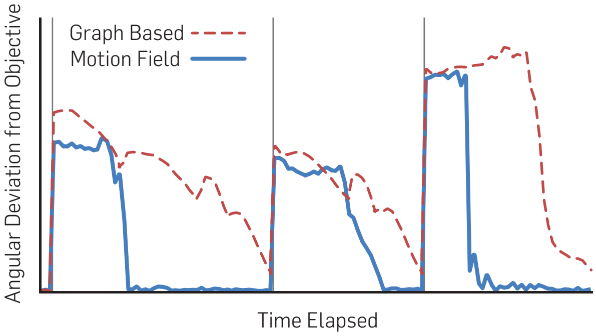


Table 1. Response times in seconds for the direction task until converging within 5 degrees of desired direction

Representation	Minimum	Average	Maximum
Graph-based	0.31	0.94	2.36
Motion field	0.21	0.40	1.01
Motion field ×10	0.21	0.49	1.23
Motion field ×20	0.25	0.66	1.19
Motion field ×30	0.38	0.78	1.93

Motion field ×10 denotes ten-fold temporally compressed value function on the motion field (N = 10). Motion field ×20 and ×30 are defined similarly. A motion field with thirty-fold temporal compression has agility similar to graph-based control, while even a twenty-fold compression is significantly more responsive than the graph-based alternative.

displacement changes. Then we measured the time until the character converged to within 5 degrees from the desired direction and 0.1 meters from the desired tracking line. We observed similar losses of agility (see Table 2).

Storage requirement and computational load. The uncompressed value function for the direction-following task is stored in 320KB. The compressed value functions required 35KB, 19KB, and 13KB for 10×, 20×, and 30× cases, respectively. This compares to the storage required for the graph-based method of 14KB. We believe this is reasonable and allows flexible trade-off between storage and agility. For more complex tasks, the size increase of the value functions are in line with the size increase for graph-based value functions.

The approximate nearest neighborhood (ANN)[12] queries represent most of the computational cost. The runtime performance depends on the sample action size k (Equation (7)), as we make $(k + 1)$ ANN calls to find the optimal action: one ANN call to find the neighbors of the current state, and then k more ANN calls to find the neighbors of the next states to evaluate value by interpolation. We believe localized neighborhood search can reduce the cost of the n subsequent calls, because the next states tend to be quite close to each other at 30 Hz.

The same ANN overhead applies at learning time. A naive learning implementation takes hours to learn a value function for a large database or a high-dimensional task. By caching the result of the ANN calls on the fixed motion samples, we can dramatically speed up learning time to just a couple minutes.

5.2. Perturbation

Because each motion state consists of a pose and a velocity, the space of motion states the character can occupy is identical to the phase space of the character treated as a dynamic system. This identification allows us to easily apply arbitrary physical or nonphysical perturbations and adjustments. For example, we can incorporate a dynamics engine or inverse kinematics. Furthermore, we do not have to rely on target poses or trajectory tracking in order to define a recovery motion. Recovery occurs automatically and simultaneously with the perturbation as a by-product of our motion synthesis and control algorithm.

To test the integration of perturbations into our synthesis algorithm, we integrated pseudo-physical interaction with motion field driven synthesis. This was done by using a physics simulator to determine how the character would respond to a push or a pull, and blending the results of this simulation into the flow of the motion field. We tested the perturbations using both passive and controlled motion fields on the following four datasets:

1. 18 walks, including sideways, backwards, and a crouch;
2. dataset 1 plus 7 walking pushes and 7 standing pushes;
3. 5 walks, 6 arm pulls on standing, 6 arm pulls on walking, 7 torso pushes on standing, and 7 torso pushes on walking;
4. 14 walks and turns.

The character responds realistically to small or moderate disturbances, even in datasets 1 and 4 that only contain non-pushed motion capture. In datasets 2 and 3 with pushed data, we observe a wider variety of realistic responses and better handling of larger disturbances. Forces applied to different parts of the character's body generally result in appropriate reactions from the character, even in the presence of user control.

We have, however, observed some cases where forces produced unrealistic motion. This occurs when the character is pushed into a state far from data with a reasonable response. This can be addressed by including more data for pushed motion.

Table 2. Response times in seconds for the line-following task until converging within 5 degrees of desired direction and 0.1 meters from the desired tracking line

Representation	Minimum	Average	Maximum
Graph-based	0.47	1.30	2.19
Motion field	0.30	0.57	1.26
Motion field ×10	0.30	0.68	1.42
Motion field ×20	0.42	0.91	2.51
Motion field ×30	0.55	1.45	3.56

In this two-dimensional control example, the twenty-fold compression is still more responsive than the graph-based control.

6. DISCUSSION

This paper introduces a new representation for character motion and control that allows real-time-controlled motion to flow through the continuous configuration space of character poses. This flow alters in response to real-time user-supplied tasks. Due to its continuous nature, it addresses some of the key issues inherent to the discrete nature of graph-like representations, including agility and responsiveness, the ability to start from an arbitrary pose, and response to perturbations. Furthermore, the representation requires no preprocessing of data or determining where to connect clips of captured data. This makes our approach flexible, easy to implement, and easy to use. We believe that structureless techniques such as the one we propose will provide a valuable tool in enabling the highly responsive and interactive characters required to create believable virtual characters.

Although the motion field representation can be used by itself, we think it can easily integrate with graph-based approaches. Since motion fields make very few requirements of their underlying data, they can directly augment graph-based representations. In this way, one could reap the benefits of graphs (computational efficiency, ease of analysis, etc.) when the motion can safely be restricted to lie on the graph, but retain the ability to handle cases where the motion leaves the graph (e.g., due to a perturbation) or when extreme responsiveness is required.

Just as with any other data-driven method, our method is limited by the data it is given. So long as the character remains close to the data, the synthesized motion appears very realistic. When the character is far from the data, realism and physical plausibility of the motion declines. Although always limited by the presence of data, more recent research has used Gaussian process latent variable models to perform well even when constructed with relatively little motion data.[9] We also expect that the range of plausible motion can be extended by an incorporation of concepts from physical dynamics (inertia, gravity, etc.) into the integration process.

More generally, we feel that motion fields provide a valuable starting point for motion representations with which to move beyond a rigidly structured notion of state. We believe that structureless motion techniques—such as ours—have the potential to significantly improve the realism and responsiveness of virtual characters, and that their applicability to animation problems will continue to improve as better distance metrics, integration techniques, and more efficient search and representation methods are developed.

Acknowledgments

This work was supported by the UW Animation Research Labs, Weil Family Endowed Graduate Fellowship, UW Center for Game Science, Microsoft, Intel, Adobe, and Pixar.

References

1. Arikan, O., Forsyth, D.A. Interactive motion generation from examples. *ACM Trans. Graph. (ACM SIGGRAPH 2002) 21*, 3 (2002), 483–490.
2. Arikan, O., Forsyth, D.A., O'Brien, J.F. Pushing people around. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2005), 59–66.
3. Chai, J., Hodgins, J.K. Performance animation from low-dimensional control signals. *ACM Trans. Graph. 24* (2005), 686–696.
4. Ernst, D., Geurts, P., Wehenkel, L., Littman, L. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res. 6* (2005), 503–556.
5. Heck, R., Gleicher, M. Parametric motion graphs. In *Proceedings of Symposium on Interactive 3D Graphics and Games (I3D) 2007* (Apr. 2007).
6. Kovar, L., Gleicher, M., Pighin, F. Motion graphs. *ACM Trans. Graph. 21*, 3 (Jul. 2002), 473–482.
7. Lee, J., Chai, J., Reitsma, P.S.A., Hodgins, J.K., Pollard, N.S. Interactive control of avatars animated with human motion data. *ACM Trans. Graph. 21*, 3 (Jul. 2002), 491–500.
8. Lee, Y., Lee, S.J., Popović, Z. Compact character controllers. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 Papers* (2009), ACM, New York, 1–8.
9. Levine, S., Wang, J.M., Haraux, A., Popović, Z., Koltun, V. Continuous character control with low-dimensional embeddings. *ACM Trans. Graph. 31*, 4 (2012), 28.
10. Lo, W.Y., Zwicker, M. Real-time planning for parameterized human motion. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008).
11. McCann, J., Pollard, N. Responsive characters from motion fragments. *ACM Tran. Graph. (SIGGRAPH 2007) 26*, 3 (Jul. 2007).
12. Mount, D., Arya, S. Ann: A library for approximate nearest neighbor searching. 1997. http://www.cs.umd.edu/~mount/ANN/.
13. Park, S.I., Shin, H.J., Shin, S.Y. On-line locomotion generation based on motion blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), ACM, New York, 105–111.
14. Shin, H.J., Oh, H.S. Fat graphs: Constructing an interactive character with continuous controls. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006), Eurographics Association, Aire-la-Ville, Switzerland, 291–298.
15. Sutton, R., Barto, A. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.
16. Treuille, A., Lee, Y., Popović, Z. Near-optimal character animation with continuous control. *ACM Trans. Graph. 26*, 3 (2007), 7.
17. Wang, J.M., Fleet, D.J., Hertzmann, A. Gaussian process dynamical models for human motion. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 2 (2008), 283–298.
18. Ye, Y., Liu, K. Synthesis of responsive motion using a dynamic model. *Comput. Graph. Forum (Eurographics Proceedings) 29*, 2 (2010).
19. Zhao, L., Safonova, A. Achieving good connectivity in motion graphs. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation* (Jul. 2008), 127–136.
20. Zordan, V.B., Majkowska, A., Chiu, B., Fast, M. Dynamic response for motion capture animation. *ACM Trans. Graph. 24*, 3 (2005), 697–701.

Yongjoon Lee (yjlee@bungie.com), Bungie, Inc., Bellevue, WA.

Kevin Wampler and Jovan Popović ({kwampler, jovan}@adobe.com), University of Washington, Adobe, Inc., Seattle.

Gilbert Bernstein and Zoran Popović ({gilbo, zoran}@cs.washington.edu), University of Washington, Seattle.

Faculty Search

ShanghaiTech University

The newly launched *ShanghaiTech University* invites highly qualified candidates to fill multiple tenure-track/tenured faculty positions as its core team in the School of Information Science and Technology (SIST). Candidates should have exceptional academic records or demonstrate strong potential in cutting-edge research areas of information science and technology. They must be fluent in English. Overseas academic connection or background is highly desired.

ShanghaiTech is built as a world-class research university for training future generations of scientists, entrepreneurs, and technological leaders. Located in Zhangjiang High-Tech Park in the cosmopolitan Shanghai, ShanghaiTech is ready to trail-blaze a new education system in China. Besides establishing and maintaining a world-class research profile, faculty candidates are also expected to contribute substantially to graduate and undergraduate education within the school.

Academic Disciplines: We seek candidates in all cutting edge areas of information science and technology. Our recruitment focus includes, but is not limited to: computer architecture and technologies, nano-scale electronics, high speed and RF circuits, intelligent and integrated signal processing systems, computational foundations, big data, data mining, visualization, computer vision, bio-computing, smart energy/power devices and systems, next-generation networking, as well as inter-disciplinary areas involving information science and technology.

Compensation and Benefits: Salary and startup funds are highly competitive, commensurate with experience and academic accomplishment. We also offer a comprehensive benefit package to employees and eligible dependents, including housing benefits. All regular ShanghaiTech faculty members will be within its new tenure-track system commensurate with international practice for performance evaluation and promotion.

Qualifications:

- A detailed research plan and demonstrated record/potentials;
- Ph.D. (Electrical Engineering, Computer Engineering, Computer Science, or related field);
- A minimum relevant research experience of 4 years.

Applications: Submit (in English, PDF version) *a cover letter, a 2-page research plan, a CV plus copies of 3 most significant publications, and names of three referees to:* **sist@shanghaitech.edu.cn** by **June 30th**, 2014 (until positions are filled). *For more information, visit http://www.shanghaitech.edu.cn.*

Lamar University
Department of Computer Science
Assistant Professor

Lamar University's Department of Computer Science seeks applications for a tenure-track **Assistant Professor** position beginning fall 2014. Applicants must have a PhD in Computer Science and a strong commitment to teaching and research. Particular areas of interest include computer graphics and vision, high performance computing, computational science and scientific computing, security and privacy, data mining and cloud computing, and bioinformatics and computational biology. Lamar is an AA/EEO state supported university of approximately 14,500 students. It offers the B.S. and M.S. in Computer Science. There are 9 full-time faculty and 260 undergraduate and graduate majors.

Apply at http://jobs.lamar.edu/postings/1499 Review of applications will begin on July 1, 2014, and continue until the position is filled. If you have additional questions, please address them at dept_cs@lamar.edu

Academic Career Opportunities

THE UNIVERSITY OF CHICAGO

Position Title: Lecturer

Req # 02129

The Masters Program in Computer Science (MPCS) at the University of Chicago invites applications for the position of **Lecturer**.

This is a three year full-time teaching position, with possibility of renewal, and involves teaching six courses across the four academic quarters of the year (Fall, Winter, Spring, Summer).

Teaching duties will involve: (1) teaching an "Immersion Programming" class for students who are entering the MS program with no prior programming experience, (2) teaching a core Programming class following the Immersion Programming class, with (3) the remaining teaching load fulfilled by teaching core and elective classes in the Lecturer's field of expertise. Candidates with a Systems background (Computer Networks, Operating Systems, Computer Architecture, etc.) will be given preferred consideration.

The successful candidate will have exceptional competence in teaching and superior academic credentials. Applicants must have a Ph.D in Computer Science or a related field at time of appointment and have experience teaching Computer Science at the undergraduate or graduate level. The selection committee may also consider candidates without a Ph.D only if they have exceptional teaching credentials and at least a masters degree in a related field.

The Masters Program in Computer Science (http://csmasters.uchicago.edu/) is a terminal MS degree in Computer Science that provides a rigorous introduction to the foundations of Computer Science, while also providing in-depth and hands-on instruction in cutting-edge and industry-driven topics, including Web and Mobile Application Development, Big Data, Cloud Computing, Data Analytics, etc. The program attracts a diverse mix of students including full-time students who are typically no more than 5 years out of college, part-time students who already work in industry, and international students.

The Chicago metropolitan area provides a diverse and exciting environment. The local economy is vigorous, with international stature in banking, trade, commerce, manufacturing, and transportation, while the cultural scene includes diverse cultures, vibrant theater, world-renowned symphony, opera, jazz and blues. The University is located in Hyde Park, a Chicago neighborhood on the Lake Michigan shore just a few minutes from downtown.

Applicants must apply on line at the University of Chicago Academic Careers website at *http://tinyurl.com/mpcs-lecturer-2014*

Applicants must upload a curriculum vitae and a one page teaching statement. In addition, three reference letters will be required.

Review of complete applications, including reference letters, will begin June 1, 2014, and continue until the position is filled.

All qualified applicants will receive consideration for employment without regard to race, color, religion, sex, national origin, age, protected veteran status or status as an individual with disability.

The University of Chicago is an Affirmative Action / Equal Opportunity / Disabled / Veterans Employer.

http://tinyurl.com/mpcs-lecturer-2014

DOI:10.1145/2602556

Peter Winkler

Puzzled
Solutions and Sources

Last month (May 2014) we posted three puzzles in which you were asked to sort several cards using three stacks on a table; you were allowed to move the top card of one stack to the top of another (possibly empty) stack, with the object being to get all the cards in their natural order stacked in the leftmost place. The catch was you could see only the top cards of the stacks and had no memory. Not included were proofs that the algorithms described in the following solutions actually work; indeed, the best way to see how they work is to take three cards (or a whole suit) from a deck of playing cards and try.

1. Three cards.

Even though it would seem not too many things can be done with just three cards, there are actually 1.64×10^{18} memoryless algorithms to try, with most not working. You need brainpower. With some reasoning and experimentation, you may find that moving the cards mostly in one direction (such as to the right, including "around the corner" from the right stack to the left stack) helps avoid looping. Moreover, you generally want to expose more cards, except perhaps in some positions from which you might win. Of the several algorithms that do work, I like the following one, as suggested by my mathematics Ph.D. student Ewa Infeld, given by these four rules in priority order:

1. Seeing 2,1,–, place 1 on 2;
2. Otherwise, seeing two cards, move one card right (around the corner if necessary) to the open place;
3. Seeing just one card, move that card to the left; and
4. Seeing three cards, move the middle-rank card to the right.

2. Any number of cards.

This same algorithm works for any number of cards, numbered 1 through n. It takes quadratic time in the worst or random case, or approximately a constant times n^2 steps.

3. With just one card in the middle.

The following algorithm was submitted by Takashi Chiba in response to a puzzle column published in Japan, and communicated to me by Ko Sakai of the Graduate School of Pure and Applied Sciences, University of Tsukuba.

1. Seeing just one card, move one card to the right;

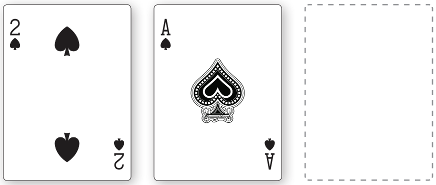
2. Seeing two cards:

If the left stack is empty, move the center card to the right stack;

If the center stack is empty, move the right card to the left stack; and

If the right stack is empty, move the left card to the right stack.

3. Seeing three cards, let the numbers on the cards be A, B, C from left to right:



If C is the largest or smallest, move the smaller of A and B onto C; and

If C is the middle, move the larger of A and B onto C.

This algorithm is cubic, requiring $(2/3)n^3 + (1/2)n^2 - (7/6)n$ steps if started with the cards stacked in reverse order on the left. Once started, it never has more than one card in the center. We thus see sorting can be done with only two LIFO stacks, a single register, and no memory. If you can beat that, please let me know…

All are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is William Morrill Professor of Mathematics and Computer Science at Dartmouth College, Hanover, NH.

[CONTINUED FROM P. 112] that if one message could causally affect another message, it would wind up with a lower timestamp. Once you have this ordering of messages, you can use it to implement a state machine. A state machine is a mathematical abstraction of a digital system as something with a state that takes inputs; an input causes it to change to a new state and generate an output. So I realized that you could implement an arbitrary distributed system by describing what it is supposed to do as a state machine, using an algorithm that implements an arbitrary state machine in a distributed system. The simplest example I could think of was to write a state machine which solved Dijkstra's mutual exclusion problem.

Several years later, you popularized the so-called Byzantine Generals Problem—in which a group of generals, some of whom may be traitors, have to reach a common decision—and invented a solution based on digital signatures.

The "Time, Clocks" paper introduced the idea of implementing a distributed system using a state machine, but that algorithm assumed no failures. Yet we build distributed systems because we expect things to fail. These days, when we talk about failures, we mean computers crashing. At the time, that body of practice did not exist, so I considered arbitrary failures, where a computer might produce a wrong answer.

Most people thought that to handle one bad computer, you had to have a total of three and use majority voting. But in a distributed system, a computer might tell one computer that the answer is 5, and the other computer that the answer is 2. So you actually need four computers.

However, if you use digital signatures, you can use three computers. If one process signs those messages, those two other processes can tell that it is faulty, because it signed messages with two different values. The original paper didn't talk about Byzantine generals; it talked about interactive consistency. But I thought this was a really important problem, and I realized that giving it this catchy notion of the Byzantine generals would make the problem more popular, so I wrote another paper with the same authors, Marshall Pease and Robert Shostak.

The Parliament of Paxos story "was a complete disaster. That paper would have died except for Butler Lampson, who understood its practical significance."

Another famous metaphor of yours is the Part-time Parliament on the fictional island of Paxos.

The people at DEC were building a distributed file system. When they explained the properties they wanted—an asynchronous algorithm that would maintain consistency despite any number of non-Byzantine faults, and would make progress if any majority of the processors were working—I thought it was impossible. So I set about trying to prove that it could not be done, and I failed by discovering the algorithm, which was independently discovered by Brian Oki and Barbara Liskov around the same time.

Yet a fully asynchronous system must either never make a mistake, but perhaps never get anything done, or else be guaranteed to get something done but perhaps make some mistakes—the so-called "safety" and "liveness" properties that you coined.

Safety properties assert that the program does not do anything bad, while liveness properties assert that it eventually does something. Paxos guarantees that, assuming only crash failures, the system never does anything wrong, but in principle it is possible for it to never succeed in doing anything. In practice, as long as the system is not behaving completely erratically, things will eventually get done.

So based on the success of the Byzantine generals, you invented the story about Paxos and the parliament.

It was a complete disaster. That paper would have died except for Butler Lampson, who understood its practical significance. Eventually, other people caught on.

Let's talk about LaTeX, the typesetting system you designed. If I understand correctly, you were writing a book yourself and were unhappy with the macros available at the time for TeX?

The system that I had been using was called Scribe. And it was really easy to use, and then TeX came along, and TeX was very hard to use, but it was also much more powerful. So my basic idea was to try to build macros that would make TeX as easy to use as Scribe.

You must have learned a lot about typography.

I think people have this sense that the purpose of typographic design is to make something look nice, whereas the real purpose is to make it easier to read.

You also introduced—and continue to refine—a powerful formalism for describing and reasoning about concurrent systems known as Temporal Logic of Actions, or TLA.

Amir Pnueli introduced temporal logic to computer science, and people quickly started using it as a specification language. I soon realized it was exactly what you wanted to use to reason about liveness.

Amir's original logic was beautiful and simple, but it was not powerful enough to express the things that we wanted to express in writing specifications.

The approach most people took was to develop more complicated temporal operators, but I realized that the best way to specify safety properties is with my old friend the state machine. And then I had the wonderful idea of extending Amir's logic not by extending the temporal operators, but by extending the class of formulas to which you apply the temporal operators. So I was able to describe the state machine in a very straightforward and practical way as a temporal logic formula, and then adding the liveness condition was just another part of the formula, so I could describe the entire specification as a single temporal logic formula, but in a practical way.

Leah Hoffmann is a technology writer based in Piermont, NY.

DOI:10.1145/2601077

Leah Hoffmann

Q&A
Divide and Conquer

Leslie Lamport on Byzantine generals, clocks, and other tools for reasoning about concurrent systems.

AFTER FOLLOWING A somewhat circuitous path to the field, ACM A.M. Turing Award recipient Leslie Lamport has spent much of his career pioneering methods to help computer scientists realize the promise of distributed systems. In industrial research positions at Massachusetts Computer Associates, SRI International, DEC, and most recently at Microsoft Research, Lamport defined problems and developed both algorithms and formal modeling and verification protocols that are fundamental to the theory of distributed systems. His solutions are as elegant as they are practical.

You majored in math at MIT, then continued in the subject in graduate school at Brandeis. When did you start working with computers?

When I studied math, the field was under the influence of a mathematician named G.H. Hardy who, as I understand it, is responsible for the view that math should be pure, abstract, and as divorced as possible from the real world. That appealed to me initially, but when I went to grad school, I began to feel differently. I then took a break and spent four years teaching math at Marlboro College, a small liberal arts college in Vermont. There, I got interested in physics and went back to grad school intending to get into mathematical physics. Meanwhile, I supported myself by working at Massachusetts Computer Associates, which everyone called Compass. That is where I wound up doing what is now called computer science. I don't think it had a name at that time.

How did you come to be involved with parallel computing?

Compass got a contract to build the FORTRAN compiler for the ILLIAC computer. The ILLIAC was a computer with 64 processors, and they needed a way for ordinary FORTRAN to be able to make use of 64 processors executing in parallel. I did the theory and the basic algorithms. I think that was probably the first thing I did that might be called real computer science.

At Compass, you read a paper by Paul Johnson and Bob Thomas that introduced the idea of using message timestamps to order the commands that are issued in a distributed system, and realized their algorithm violated causality. The resulting paper, "Time, Clocks and the Ordering of Events in a Distributed System," is still among the field's most-cited.

In that paper, I introduced a causality ordering, in which one event precedes another if it can causally affect it. I thus modified their way of keeping clocks so [CONTINUED ON P. 111]

PHOTOGRAPHS BY RICHARD MORGENSTEIN

SIGCOMM
CHICAGO 2014

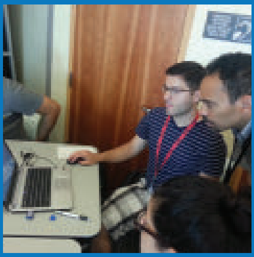
Get the Latest Software and Hardware for your Classroom

Intel® Software Academic Program benefits include:

- Global Community of Peers
- Access to Hardware
- Technical Support
- Course Material
- Software Tools
- Code Samples

Intel® RealSense™ Technology

Internet of Things



Hack-a-thons

"Intel has a long history of academic collaborations in high performance computing and semiconductors. We have expanded our university outreach to Security, Mobile Computing, Perceptual Computing and the growing Internet of Things."

Michael A. Smith Ph.D., Software Academic Program Director

Join today at www.intel.com/software/JoinAcademics

Intel® Developer Zone