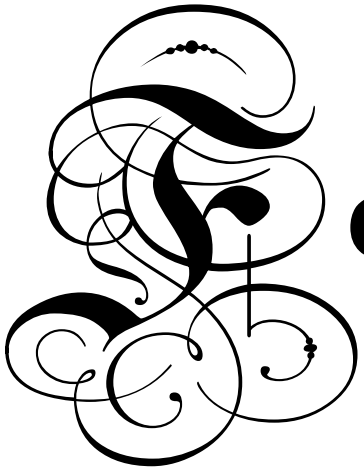


Festschrift

for Rick Rashid
on his 60th birthday

Rich Draves
Pamela Ash

9 March 2012

Dan Hart illustration for School of Computer Science,

Introduction

Rich Draves

You're holding a *Festschrift* honoring Rick Rashid on his 60th birthday. You may well ask, "What is a Festschrift?" According to Wikipedia, a Festschrift "is a book honoring a respected person, especially an academic, and presented during his or her lifetime."

The Festschrift idea dates back to August 2011, when a group of friends and colleagues suddenly realized that Rick's 60th birthday was rapidly approaching. In our defense, I can only point to Rick's youthful appearance and personality. Peter Lee suggested that we organize a Festschrift. The idea was foreign, but everyone immediately recognized that it was the right way to honor Rick.

The only problem was timing. Fortunately, this is a common problem and the belated Festschrift is somewhat traditional. We set a goal of holding a colloquium on March 9, 2012 (in conjunction with Microsoft Research's annual TechFest) and producing a volume by September 2012, within a year of Rick's birthday.

This volume has content similar to the colloquium, but some authors made a different written contribution. The DVD in the back of the book contains a recording of the colloquium.

I want to emphasize that we're celebrating Rick's career and accomplishments to date but we are also looking forward to many more great things to come. In other words, we need not fear his retirement. I remember Rick saying once that someone would need to "pry his cold dead hands off the keyboard," so I assume that Microsoft Research is not in imminent danger of losing Rick.

Many people were involved in producing the colloquium as well as this volume. Peter Lee deserves credit for originally suggesting the idea and working with me on the organization—he had some great advice. I want to thank all the contributors and all the people who came

from out-of-town for the colloquium. When you are planning something like this, you never know what you are going to get and everyone came through amazingly. Henry Honig and Daniel Betcher worked miracles at the last minute to edit videos of Nathan Myhrvold and Avie Tevanian, who could not attend the colloquium in person. Jamie Rifley, Jenifer Carlson, and Melissa Kelly provided great administrative support for the colloquium. Lynn Powers jumped in to help produce this volume and dedicated many hours to research options and review the design, along with Ann Paradiso and Tony Carbary. Pamela Ash worked full-time for several months on copy editing, layout, and the many other tasks necessary to produce a book. Thank you all!

Rich Draves

Contents

Rich Draves
Introduction i

Faculty and Staff

Raj Reddy
Educating a Child Machine 9

Mahadev Satyanarayanan
On Microkernels and Virtual Machines: The Legacy of Mach 21

Alfred Spector
A Letter 31

Jonathan Chew
How I Came to Work for Mach's Fearless Leader 33

Students

Avadis Tevanian, Jr.
Evolution of the Mach Operating System 39

Michael Young
A Letter 47

David L. Black
A Letter 49

DH-CHAP: Diffie-Hellman Enhanced CHAP for iSCSI 51

Michael B. Jones
Matchmaker: An Interface Specification Language
for Distributed Processing 67

Bill Bolosky
The Tiger Project 79

Rich Draves
Twenty-Five Years with Rick Rashid 83

Galen C. Hunt
The Drawbridge Library OS: Windows as an Application 87

Joe Barrera
A Letter 117

Founding Microsoft Research

Bill Gates
Letter to Rick Rashid 123

Nathan Myhrvold
Anecdotes from MSR's Founding 125

Historical Memo Proposing MSR 131

Dan Ling and Jack Breese
A Short History of Microsoft Research 153

Linda Stone
A Letter 163

James Larus
Software Development and Engineering Research
in Microsoft Research, Redmond 165

Ed Lazowska
Rick Rashid, Microsoft Research, and Academia 171

Xuedong Huang
Rick Rashid's Impact on Microsoft Product R&D 175

Managing Microsoft Research

Roy Levin
On Software Principles and Research Management 181

Hsiao-Wuen Hon
Rick Rashid & Microsoft Research Asia 189

Andrew Blake
Moving in Mysterious Ways 201

Christian Borgs and Jennifer Chayes
From Phase Transitions to Network Algorithms:
A Personal Research Journey on Rick's Long Leash 205

Peter Lee
The Importance of Dog Food 217

Rakesh Agrawal
The Future of Textbooks 221

Rick Rashid, Renaissance Man

Gordon Bell with Jim Gemmell
And Now, the Next 20 Years 233

Andrew Herbert
Rick Rashid—Aeroplane Pilot 239

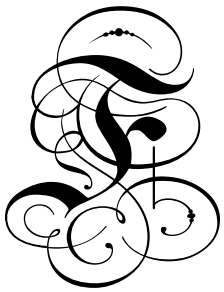
Greg Bear
Randomness, Science, and the Library of Babel 241

Terri Rashid
Seven Habits of the Highly Effective Rick Rashid 249

Photographs

Early Years 260

Recent Times 263



aculty and Staff

Educating a Child Machine

The Unfinished Agenda of Human-Level AI

Raj Reddy

March 9, 2012
Talk at Rick Rashid's Festschrift

Several eminent scientists from Turing to McCarthy to Kurzweil have made serious proposals for achieving human-level AI (as opposed to creating useful systems that act intelligently) and yet so far, there has not been a sustained and comprehensive effort towards this goal. In this paper, we present some of the past proposals and their limitations; reasons for optimism given the dramatic results of the last decade in search, speech, language understanding (translation and question answering) and autonomous navigation; and a possible roadmap for making fundamental progress towards the goal by first building a Child Machine that is teachable and can learn to reach the abilities of a 4 year old.

I. Introduction

When Rick came to CMU in 1979, it seemed that he might spend time doing vision and AI research as a continuation of his thesis. Fortuitously, I had just received a major DARPA grant for research in Distributed Sensor Networks and it was clear that we needed a distributed operating system before we could start the rest of the research. Thus began Rick's work on Mach and eventual arrival at Microsoft. Nevertheless, given his broad set of interests and contributions to AI, I thought it might be fun to layout a research plan that he could lead when he retires from his current job! It had to be challenging enough to attract his attention and one he might pull off given another 20 years of large scale effort at MSR. Hence the topic for today, namely, "Educating a Child Machine: The Unfinished Agenda of Human-Level AI."

John McCarthy once said, "Human-level AI might require 1.7 Einsteins, 2 Maxwells, 5 Faradays and 0.3 Manhattan Projects, the project coming after the conceptual breakthroughs." So

far, we neither have all the needed conceptual breakthroughs, nor the long-term infusion of billions of dollars of funding to realize human-level AI. I believe John is mostly right, albeit the needed conceptual breakthroughs may not happen in a vacuum and will need the context of a sustained long-term project. Only MSR, the world's largest computer science research group, can probably pull it off; perhaps as part of a long-term coordinated national effort and with some help from the academic community.

II. Past Proposals for Achieving Human-Level AI (HLAI)

So what do we mean by human-level AI? Here we will not concern ourselves with philosophical arguments that "machines cannot have a mind" or the pragmatic view that "the primary mission of artificial intelligence research is to create useful systems that *act* intelligently."

Several eminent people starting with Turing have attempted to answer the question of how a machine might think and exhibit human-equivalent functional behavior. Nilsson (2009) provides a review on steps, "Towards Human Level Intelligence." Here we provide a short commentary on some of the relevant proposals.

Turing (1950) starts with the question, "Can machines think?" and proposes an imitation game to resolve the question. What is less well-known is that he also proposed creating a "Child Machine" which is then subjected to an "appropriate course of education," requiring us to develop "the child programme" and "the education process" with a mechanism for learning from input and having its output corrected; by repeated give-and-take sessions between the machine and teacher. Interestingly, Turing seems to assume that HLAI need not be provided eyes or limbs!

Newell and his students (Laird et al., 1987) created and demonstrated the SOAR architecture (historically, SOAR stood for State, Operator And Result), designed to demonstrate a "unified theory" of general intelligence behavior. Over the years, Newell had evolved a set of principles such as, "there is a single elementary representation for declarative knowledge," "control in general intelligence is maintained by a symbolic goal system," "production systems are the appropriate organization for encoding all long-term knowledge," etc., which are embodied in SOAR. SOAR limits itself to cognitive tasks, and perceptual and motor tasks are left to future implementations. It is not at all clear that the architecture of SOAR would be suitable for perceptual and motor tasks or a single "unified theory" would lead to HLAI. Unfortunately, after Newell, with no one else pursuing this line of research, not much progress has taken place.

McCarthy (1999) in his classic paper, "The Well Designed Child," suggests that a Child Machine (that can learn from experience just as a human child does) should be endowed with important innate capabilities, rather than a blank slate. He proposes that independent of what a human baby might posses at birth, we should try to give the Child Machine "all we can give it." His paper proposes several possible innate capabilities and ways of evaluating their effectiveness.

Kurzweil (2005) proposes to create Super-Human AI (SHAI) by reverse engineering the human brain using nano-technologies and making it run a thousand times faster than the

human brain does. In his book on singularity, he lays out arguments why such an outcome is inevitable in 30 to 40 years. Whether his predictions are realized or not will certainly depend on the conceptual breakthroughs needed to create the HLAI (or SHAI). On this issue, he appears to assume that reengineering the brain will leave the intelligence architecture intact!

Nilsson (2005) proposes that to be considered "human-level artificial intelligence," it must be capable of automating most of the tasks that humans perform for pay. He provides a list of such tasks ranging from "Maid and Housekeeping Cleaner" to a "Computer Programmer." Rather than work toward this goal of automation by building special-purpose systems, he recommends the development of general-purpose, educable systems that can learn and be taught to perform any of the thousands of jobs that humans can perform using a "Child Machine" a la Turing with the ability to acquire the skills through learning by doing and learning from examples.

Minsky (2006) in his book, "The Emotion Machine," proposes that human-level AI will not be achieved by a single "unified theory" but would need to use multiple methods to represent knowledge, multiple ways to make inferences and multiple ways to learn. Using a "Causal Diversity Matrix," each problem-solving technique is matched to relevant problem domains: "When should we use logical reasoning?" "When should we use statistical reasoning?" "When should we use case-based reasoning?" etc.

Interestingly, almost all of the proposals stop short of design or implementation of HLAI! Newell et al. attempted to implement and demonstrate a working prototype of SOAR, but it was limited to a few cognitive tasks and has been discontinued without Newell providing the direction and drive. No plans exist for making any systematic progress and no AI group is seriously exploring the possibility of creating HLAI. If this situation continues, we will never have HLAI. We suggest that the time for making a serious attempt towards HLAI is now.

All the "gurus" above appear to have a blind spot, possibly resulting from their environment and culture. To them, a thinking machine would have capabilities similar to their own. Turing hopes that machines will eventually compete with men "in all purely intellectual fields." McCarthy assumes that a "Well Designed Child" can handle non-monotonic reasoning and introspection. Nilsson assumes that a system with human-level AI should be able to do many of the "jobs" at which people are employed. Newell and Simon thought that AI systems should be able to play chess and prove theorems!

This raises the question of whether a billion plus illiterate people in the world do or do not possess human-level intelligence. Many illiterate people can drive; use speech, language and vision; and do path planning every day. We now know that these are very challenging AI tasks, perhaps harder than the purely intellectual tasks. And even people at the "bottom-of-the-pyramid" are good at doing them!

III. Recent Landmark Accomplishments in AI

In the last decade, we have begun to see some significant breakthrough in tasks long regarded as grand challenge problems in AI. Here we will give a short description of each of these events.

Computer Chess. The first celebratory success for AI came in 1997 when Deep Blue beat Kasparov, the reigning world chess champion, and won the Fredkin Prize (established in 1979) sooner than was generally expected. Beginning with the Newell, Shaw and Simon Chess program in 1958, chess had a long history as the drosophila of AI. At the 1997 AAAI awards ceremony, many pioneers of computer chess were given a Newell Medal for Research Excellence including Richard Greenblatt, Hans Berliner, Ken Thompson, and Atkin and Slate. Their earlier contributions were deemed to be essential stepping stones for the ultimate success of Deep Blue. (Hsu 2002; Anantharaman et al. 1990; Berliner et al. 1990; Campbell et al. 1999).

Autonomous Navigation. The dramatic demonstrations of "driverless vehicles" in 2005 and 2007 were landmark events in the history of AI. In a competition sponsored by DARPA, vehicles in the 2005 race (won by Stanford's Stanley) had to cover 132 miles of off-road course including three narrow tunnels and navigate more than 100 sharp left and right turns. Vehicles in the 2007 race (won by CMU's Boss) had to drive in an urban setting, obeying all traffic regulations while negotiating with other traffic and obstacles and merging into traffic. Autonomous vehicle research had been funded by DARPA since 1984 (much of it at CMU) and had demonstrated different levels of performance over the 2 decades prior to the Grand Challenge races, confirming the maxim that breakthroughs don't happen in a vacuum! (Thrun et al. 2006, Urmson et al. 2007). Other autonomous systems like Robosoccer have made substantial progress but are not yet at human level of performance (Veloso, 2003).

Language Understanding. Over the last decade we have seen several breakthroughs in language understanding in the areas of information retrieval, translation and question answering. All of these were previously thought to be infeasible.

> ***Information Retrieval.*** Since Lycos was developed by Fuzzy Mauldin at CMU in 1994, full text search engines have become a reality. By now, daily use of search engines has become ubiquitous and companies that started the wave like Yahoo and Google have become commercial miracles. Google today has over $40B in revenue and $200B market cap. Bing, Google and Yahoo have emerged as major technology providers and innovators. It is interesting to note that Lycos had its origins from DARPA funding at CMU and Google's origins come from NSF funding of digital library research at Stanford. In spite of significant investment of resources, these systems still make some mistakes because of incomplete models for understanding of human language. (Croft et al. 2009)
>
> ***Language Translation.*** Google Translate is able to produce tolerable translation, deemed to be impossible a decade ago, from any pair of 60+ languages. CMU established the Machine Translation Center (later called Language Technology Institute) in 1985, and did much of the seminal work on rule-based and example-based MT. However, at best, quality translation was possible only for carefully authored manuals (and other such industrial documents) and not for freeform found text. IBM later pioneered a statistical approach and achieved comparable results. The best

commercially available tool, SYSTRAN, worked on a handful of languages and not always bidirectional. By 2006, Google was able to outperform all other systems and was ranked first in NIST evaluations in 2009. Google Translate's power comes not so much from particularly complex statistical methods, but rather from access to very large data sets and computational power. Recently, some hybrid systems combining knowledge and statistics seem to do somewhat better, following the maxim, "a little knowledge goes a long way." (Google Translate, 2011)

Question Answering. In 2011, IBM (with a little help from CMU) demonstrated a question answering system capable of competing with humans on the TV show Jeopardy, and winning. Again, it was a feat deemed to be impossible until it was demonstrated. With roughly 3 seconds to answer each question, and a fraction of a second to decide whether it can answer the question, the task required incorporation of many other criteria such as precision, confidence estimation, clue selection, and betting strategy. Whereas most search engines merely produce a set of links for you to review and formulate your own answer, a QA system requires integration of information from multiple sources (or links) and then creates an answer within a 3-second response time. It is interesting to note that Watson's origins arise from long-term DARPA funding of Natural Language Processing research and IARPA funding of AQUAINT research at IBM and several universities, as a team project. (Ferrucci et al. 2010)

Speech Understanding. The 2011 release by Apple of the iPhone 4S with the SIRI spoken language understanding system represents a landmark event. Prior to that event, it was believed that recognition of "unrehearsed spontaneous speech" in an uncontrolled environment was nearly impossible. Before that, Microsoft and Google demonstrated respectable performance in voice-search and TellMe pioneered in the use of speech and dialog systems in call center applications. While similar systems were demonstrated in academic laboratories in the 1990s, use by the open population was known to be an order of magnitude more difficult. It is interesting to note that SIRI's origins arise from DARPA funding of CALO research at SRI and several universities as a team project. (SIRI, 2011)

Missing Science. In each of the above tasks, systems still use incomplete models and take many shortcuts to get the job done. Given that almost 90% of the human brain is engaged in processing and responding to sensory-motor stimuli, we can expect to take many years to get to HLAI.

What all of these breakthroughs have in common is the access to unlimited data and unlimited computation which became possible in the last decade. We have had a million-fold improvement in memory, bandwidth and computation over the last 30 years. Suddenly, previously infeasible algorithms and approaches became possible. All of a sudden, brute force seems to be an acceptable approach to AI and problem solving. Perhaps the human brain with access to 1013 neurons also uses brute force strategies for recognition and recall.

Interestingly, almost all of these breakthroughs had their beginnings at CMU and other academic centers, but it took the resources of industry to bring about deployable working systems. Along the way, intermediate results have spawned billion dollar start-ups and can be expected to continue to lead to numerous new start-ups.

IV. A Roadmap for Creating Human-Level AI (HLAI)

Most proposals for creating human-level AI do not provide a constructive plan on how to go about it. We propose to create an architecture for a "Child Machine" that can learn and is teachable a la Turing and McCarthy. HLAI would have a mechanism for learning from input and having its output corrected, and by repeated give-and-take sessions where the machine would be presented with stimulus and produce a response in kind, which is instantaneously corrected as needed. We assume that the Child Machine would have the full range of sensory-motor capabilities (until shown to be difficult or costly) of a human child and would acquire perceptual and motor skills through learning-by-doing and learning-from-examples, initially leading to a functional equivalent of a 4 year old. (See also Brooks 2008)

The functionality we seek for HLAI is that which can be expected to be possessed by MOST humans on the planet. We start with the premise that all humans on the planet posses human-level intelligence (HLI), whether they can read or not, do mathematics and sciences or not, or even whether they can be taught employable job skills or not. Even illiterate people on the planet can learn to drive, use speech, language and vision; and do (an albeit non-literate type of) planning, problem solving and abstract thinking every day. This is the functionality we seek for HLAI initially, to be achieved not by programming, but by permitting the Child Machine(s) to inhabit, learn, discover and survive in a controlled environment.

We propose to sub-divide the goal of creating HLAI into sub-goals which can be studied in isolation as manageable chunks. The hope is that each level provides the foundation and building block for the next level. However, each level can be studied independently without waiting for the preceding levels to be fully working by using humans to provide equivalent simulated functionality.

The architectures discussed in McCarthy's, "The Well Designed Child" and Minsky's, "The Emotion Machine" spend considerable time on the internal structure and function of an HLAI system. Here, instead, we discuss only what an HLAI should be able to do (the external functionality of HLAI) and not what is inside the "black box." Future system designers should have the flexibility to pursue whichever design gets the job done. In the process, we should be prepared to toss out a system and start anew as needed, much like Einstein did with Newtonian mechanics. But the new system should subsume the functionality of the old systems.

I propose that we start with Piaget's stages of cognitive development (and adapt as needed) to arrive at the functionality we seek from an HLAI system. We divide the HLI and HLAI into 7 stages (Level 0 to Level 6) that approximately correspond to Piaget's stages. We assign to each of them functionality exhibited by Humans (as discussed in developmental psychology

literature). Up to level 2, we discuss capabilities every person on the planet might be expected to have, literate or not. Subsequent levels will require HLAI to learn and discover from experience (as may be in the case of an illiterate villager) or attend school and be taught new tools and techniques by a teacher.

Research Strategy for HLAI

It is clear that we don't know how to build a CM (Child Machine) or how to educate it. As Einstein said, "If we knew what it was we were doing, it would not be called research, would it?" That is why we need a large scale national effort to build a series of prototypes to make progress towards the goal of HLAI. Given below are some starting assumptions (which may be modified as needed with experience).

We define multiple levels of HLAI, many of which would be the levels of intelligence most people on the planet would possess, starting with creating a Level 0 Child Machine (L0CM) at birth with Level 7 Child Machine (L7CM) modeling the abilities of people with post secondary education. Once we have a system that is educable at the fundamental levels of 1 and 2, it is assumed the remaining levels would not be much more difficult.

What about geniuses and super intelligent people? We can assume that such people result from a rare or unusual genetic mutation. In many cases, such unique attributes may go undetected and/or treated as misfits when not properly understood. Conditions such as autism also seem to lead to unusual capabilities. In this paper, we suggest that we wait until we have created the HLAI possessed by average people and treat others as exceptional cases. If Kurzweil is right and we end up creating SHAI, then we can all become SHAI, not just a select few.

In developing HLAI, we should follow a "fail-fast strategy" of rapid prototyping and iteration. Rather than spend a lot of time in getting the design perfect (like NASA is prone to do), we should adopt a "fail-fast" strategy routinely used these days by software developers, variously called extreme programming, agile programming, scrum, etc.

In research, unlike conventional software development, we often don't know the correct architecture or the component definition. We anticipate that many different ideas and approaches have to be explored before settling on one or a few. Thus, we should use multiple teams and/or multiple approaches towards each goal. However, we should be able to reuse components or pieces of code that work (sound like evolution?) from any or all approaches.

L0CM: Level 0 Child Machine (at Birth): Architecture for learning, goal-directed behavior, survival and emotion.

At this level, we seek to provide the HLAI, the functionality of a newborn at birth, or possibly second trimester. We might call this the basic hardware level, giving the system innate capabilities of assimilation and accommodation (Just, M., et al. 2007 and 2010).

For example, if the L0CM architecture is comprehensive, then, after being exposed to 6 months equivalent sensory data that a human child would have been exposed to, an L0CM should

smile (or equivalent) upon observing mother's image in L0CM's visual field. And after being exposed 10 months equivalent sensory data, an L0CM should begin to utter "ma" sound among other babble.

Learning architecture should enable memory structures to store, match, recognize, retrieve, change, index and organize sensory-motor data. We assume the availability of unlimited memory, computation and bandwidth so that brute-force solutions (that have worked so far) become accepted features of architecture for intelligence. Signal processing, image processing and feature extraction for indexing, matching, retrieval, etc. can be expected to be an integral part of the capabilities of the L0CM.

At Level 0, we also need to include "interrupt" features for goal directed behavior, survival (e.g., HLAI equivalent to pain and hunger) as well as architecture for emotions such as joy, grief, fear, envy, greed, anger and need. Usually, most proposals do not consider survival and emotion to be part of HLAI, but are included here because it may be an essential part of appearing to be intelligent.

Hardware for L0CM. For now we should assume that we are not limited by biological constraints. The Hardware for L0CM can be assumed to have unlimited memory, computation and bandwidth.

Software for L0CM. L0CM is primarily a sensory signal processing and learning system. It should have mechanisms for feature extraction from signals, for learning and having its output corrected; by repeated give-and-take sessions between the machine and teacher.

L1CM: Level 1 Child Machine (ages 0-1): Architecture for Vision, Hearing and Motor Processes

At Level 1, we seek to provide HLAI with functional equivalents to vision, hearing and motor processes. Here we seek to demonstrate that the Level 0 architecture is adequate to educate HLAI to acquire, process, index, recognize and adopt day to day images and sounds of family and the environment, without explicit programming of the functionality. At Level 1, we should be able to observe the mechanisms for learning from input and having its output corrected, in interactive sessions between the machine and teacher. We only assume that the system has access to same type sensory-motor data that a human child can be expected to have and is able to self-organize the material.

Emotional responses to denial (or satisfaction) of goal-directed behavior including possibly smell and taste (will the child machine eat?) should be observable. Movement, vocalization and related motor functions in response to local stimuli should also be observable.

At this level, we are attempting to develop and observe stimulus/response behaviors typical of a child and associate visual, tactile and motor representations of objects and for the child

machine to discover by itself the concepts of object permanence (the understanding that objects are solid, permanent, and continue to exist when out of sight).

Educating L1CM. At L1, we assume a CM is able to acquire, use, and recognize images, sounds, and motion and exhibit goal directed behavior. The assumption here is that any unsatisfied expectation or goal would result in an emotional response and represents the survival instinct. A CM may not need the same instincts as a human, but should be able to recognize changes and dangers in their environment. While much of the behavior could be the stimulus-response type of "recognition and recall" behavior, we also need to embed "interrupt-driven" goal directed behavior based on continuous monitoring of external conditions and extraneous situations.

We start with the assumption that we educate the CM to learn from "birth" by exposing HLAI to the same sensory experiences that a human child is being exposed to. We know that as children grow, their past experiences will shape who they are, and how they perceive the world. This means that we should select a wide diversity of child rearing environments, initially limiting US locations. The choice can be based on socio-economic status, language and cultural differences. The resulting diversity among HLAIs should be interesting and can be a source of new research in itself.

Initially, we should select and instrument 5 to 10 environments where an infant child is growing up (unscripted reality TV?) and collect every type of sensory data that each child is being exposed to, including images, sounds, smells, touch, pressure, light and temperature. All this data is continually streamed to the HLAI Research Center, where different CMs are exposed to different data sets. CMs are expected to process, analyze and recognize patterns using built-in innate capabilities. Initially, all CMs are assumed to be endowed with the same innate capabilities, although this choice could itself be a part of the research experiment design.

Habituation and recognition come early. Using these mechanisms, whenever a new image is received, it would be matched with previously received images to either "index and store as a new image" or "just another instance of known image." We know that infants only begin to acquire concepts of object identity around the age of 6 months (e.g., mother's face) and of object permanence after the age of 18 months. Similarly, an L1CM should be able to understand and generate one word utterances by 12 months and short phrases by 24 months.

We need not be concerned with the exact timetable of a human child, but we need to ensure images, sounds and other sensory data are being matched and recognized as known or unknown by L1CM, and appropriate action taken for indexing, storage and retrieval as well as linking multi-sensory representations of the same object and its meaning.

L2CM: Level 2 Child Machine (ages 2-3): Architecture for Speech and Language Acquisition

At Level 2, the primary goal is to provide HLAI with functional equivalence of speech and language understanding and spoken language output. The HLAI is exposed to several thousand hours of raw speech and language data with possible corrective action by the designer.

> ***Educating CM for L2 performance.*** At L2, we assume a CM is able to acquire, use, recognize and generate speech and language equivalent to a 2 to 4 year old human child recognizing and generating full sentences possibly with few errors and a vocabulary of around 1,000 words and phrases. L2CM is provided the same raw data that the human child is exposed to including clarification and correction by a teacher.

L3+CM: Level 3+ Child Machines

For Level 3 to 6 below, we state the broad functionality without providing any guidance or commentary.

Level 3 (ages 4-5)	Architecture for Reading and Writing
Level 4 (ages 6-10)	Architecture for Primary School Level Competence
Level 5 (ages 11-13)	Architecture for Elementary School Level Competence
Level 6 (ages 14-17)	Architecture for Secondary School Level Competence
Level 7 (ages 18+)	Architecture for Post-Secondary Level Competence

The only requirement is to exhibit the functionality typically demonstrated by an illiterate person for problem solving, logical thinking and abstract thinking should also be achievable by an HLAI machine, without having to attend school as implied at each of the levels above. This will help us distinguish between discovered knowledge versus taught knowledge of HLAI.

The approach outlined above should result in a clearer understanding of the missing science, unsolved problems and their apparent complexity within 5 to 10 years, akin to our knowledge on understanding speech towards the end of DARPA speech understanding systems project in 1976 (Newell, 1971; Reddy, 1976). The fact SIRI emerged 40 years later, while gratifying, was not knowable at that time.

V. Conclusion

In conclusion, my main thesis is that human-level AI should be achieved by teaching a "Child Machine" as Turing and McCarthy have suggested. The main difference from other proposals is that the baseline for HLAI should be the tasks that almost everyone on the planet could do and not just educated people. This paper suggests that HLAI should be achieved by organizing research into incremental stages of increased functionality, roughly analogous to Piaget's theory of cognitive development. We present one such roadmap for progress starting with Level 0 HLAI, that which every human in the world can be expected to posses at *birth*, with subsequent levels slowly escalating capability requirements.

human-level AI will never be realized unless we have sustained long-term effort towards that goal. We need a large scale sustained funding and effort (100 to 1000 PhDs?) and we need breakthrough ideas. These new ideas will not sprout in isolation but within the context of a large national project with clearly stated research objectives.

As I scan the world, there is only one place that can provide leadership to a national effort and has the necessary critical mass of computer scientists, namely Microsoft Research Labs. It seems to me that Rick needs something challenging to do when he retires from his current job and this would keep him busy for the next 20 years!

Good luck and Godspeed, Rick!

VI. Acknowledgement

I would like to thank Jaime Carbonell, Ed Feigenbaum, Marcel Just and Nils Nilsson for comments and input on the paper.

VII. References

Anantharaman, T.; Campbell, M.; and Hsu. F. 1990. Singular extensions: Adding selectivity to brute-force searching. Artificial Intelligence, 43, 99-109.

Berliner, H.J.; Goetsch, G.; Campbell, M.S.; and Ebeling, C. 1990. Measuring the performance potential of chess programs. Artificial Intelligence, 43, 1, 347-367.

Brooks, Rodney. 2008. I, Rodney Brooks, Am a Robot. IEEE Spectrum, 45, 6, 62-67.

Campbell, M, Hoane, J, and Hsu, F. 1999. Search Control Methods in Deep Blue. AAAI Spring Symposium 1999.

Croft, B., Metzler, D., Strohman, T. 2009. *Search Engines: Information Retrieval in Practice*. Addison Wesley, New York.

Ferrucci, D. et al. 2010. Building Watson: An Overview of the DeepQA Project. AI Magazine, 31, 3, 59-79

Google Translate. 2011. http://en.wikipedia.org/wiki/Google_Translate

Hsu, F.-H. 2002. *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton, NJ: Princeton University Press.

Just, M. and Varma, S. 2007. The Organization of Thinking: What Functional Brain Imaging reveals about the Neuroarchitecture of Complex Cognition. *Cognitive, Affective and Behavioral Neuroscience,* 7, 3, 153-191

Just, MA, Cherkassky, VL, Aryal, S, Mitchell, TM. 2010. A Neurosemantic Theory of Concrete Noun Representation Based on the Underlying Brain Codes. PLoS ONE, 5, 1, 1-15

Kurzweil, Ray. 2005. *The Singularity Is Near: When Humans Transcend Biology*. Viking Press, New York. See also Spectrum special issue http://spectrum.ieee.org/static/singularity

Laird, John; Newell, Allen; and Rosenbloom, Paul 1987. SOAR: An Architecture for General Intelligence. AI Journal, 3, 1, 1-64.

McCarthy, J. 1999. The Well-Designed Child. Unpublished memo, available at http://www-formal.stanford.edu/jmc/child.pdf

Minsky, Marvin. 2006. *The Emotion Machine*. Simon and Schuster, New York. See also Minsky, M.; Singh, P.; and Sloman, A. 2004. Designing Architectures for Human-Level Intelligence. AI Magazine 25, 2,113–124

Newell, A. et al. Speech Understanding Systems. 1971: Final Report of a Study Group. (Reprinted by North-Holland/American Elsevier, Amsterdam, Netherlands, 1973).

Nilsson, Nils J. 2005. Human-Level Artificial Intelligence? Be Serious!, AI Magazine, 26, 4, 68-75

Nilsson, Nils J. 2009. *The Quest for Artificial Intelligence*. Cambridge University Press, New York.

Reddy, D.R. 1976. Speech Recognition by Machine: A Review. Proceedings of IEEE, 64, 4, 501-531.

SIRI. 2011. http://en.wikipedia.org/wiki/Siri_(software)

Thrun, S.; Montemerlo, M.; Dahlkamp, H.; Stavens, D.; Aron, A.; Diebel, J.; Fong, P.; Gale, J. et al. 2006. Stanley: The Robot That Won the DARPA Grand Challenge. The Journal of Robotic Systems, 23, 9, 661-692.

Turing, A. M. 1950. Computing Machinery and Intelligence. Mind, LIX, 236, 433–460

Urmson, C. et al. 2007. Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge. Robotics Institute Technical Report, Carnegie Mellon University.

Veloso, Manuela. 2003. Autonomous Robot Soccer Teams. The Bridge, National Academy of Engineering, 33, 1, 8-12

On Microkernels and Virtual Machines: The Legacy of Mach

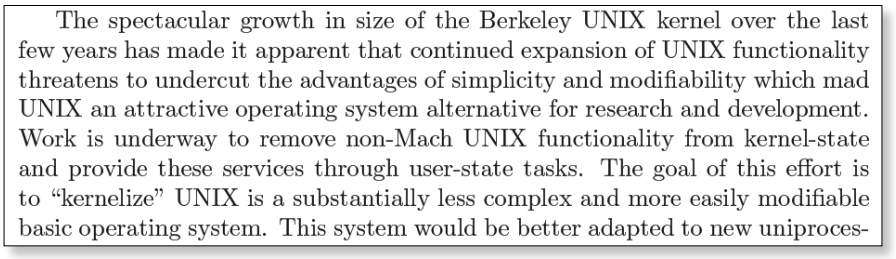
Mahadev Satyanarayanan

School of Computer Science
Carnegie Mellon University

During the early years of Mach, anyone who talked to Rick Rashid soon encountered two ideas that excited him. One was the idea that operating systems had grown too large and complex, and that the time had come to simplify and shrink. The second was the idea that different operating systems offered the same core functionality underneath a lot of superficial diversity, and it should therefore be possible to dynamically choose which guise to interact with. Although these two powerful ideas are interrelated in the Mach implementation, it is useful to think of them as distinct concepts from a historical perspective. In the roughly two decades since active research on Mach ended, how have these concepts fared in the marketplace of ideas? What is their intellectual legacy for OS research? Does anyone still care about them? We examine these issues in this brief note.

2. Simplify and Shrink

Figure 1 is an excerpt from the earliest publication I found that mentions the microkernel concept in Mach. By the mid-1980s, the dominant operating system used by academic researchers (Unix) had grown from a small, tightly organized and easily understood code base into a large and unwieldy body of code that was difficult to understand, debug and maintain. Some of the growth was due to introduction of new functionality such as support for virtual memory and support for networking in the form of the socket interface. Also contributing to size and complexity was the need for portability of the code across many different hardware platforms. The net impact on complexity was multiplicative, not just additive. The Mach



The spectacular growth in size of the Berkeley UNIX kernel over the last few years has made it apparent that continued expansion of UNIX functionality threatens to undercut the advantages of simplicity and modifiability which mad UNIX an attractive operating system alternative for research and development. Work is underway to remove non-Mach UNIX functionality from kernel-state and provide these services through user-state tasks. The goal of this effort is to "kernelize" UNIX is a substantially less complex and more easily modifiable basic operating system. This system would be better adapted to new uniproces-

FIGURE 1: Earliest Mach Reference (1986) to the Microkernel Concept [1]

manifesto proposed a fundamental restructuring of an operating system into a small, simple core combined with mechanisms to easily extend that core.

The importance of extensibility in operating system design had been identified as early as 1970 by Per Brinch Hansen [9]. His design rationale for the RC 4000 operating system stated: "... the main problem in the design of a multiprogramming system is not to define functions that satisfy specific operating needs, but rather to supply a system nucleus that can be extended with new operating systems in an orderly manner." This fundamental insight into operating system structure was lost in the explosive growth of computing hardware and software in the 1970s and early 1980s.

Mach's rediscovery of this attribute came at an opportune time. Its push towards simplicity and extensibility coincided with the emergence of workstation and server hardware with multiple processors. The monolithic structure of Unix, especially its coarse-grain concurrency control around kernel entry and exit, muted the performance benefits of using multiple processors. Introducing mechanisms for finer-grain concurrency control within the kernel was a difficult task, most likely requiring an entire rewrite of the kernel. Such a rewrite was an opportune time to make fundamental changes to the structure of Unix.

By the late 1980s, Mach software releases of growing portability and efficiency were available for a diversity of hardware platforms, some with multiple processors. Initial releases (Mach 2.5 and 2.6) retained the monolithic structure of BSD Unix, but reengineered the design of the virtual memory system, kernel locking mechanisms, and related components. Later in the evolution, starting with Mach 3.0, began the process of fundamental restructuring into a true microkernel architecture.

Accompanying the successful software releases were a number of scholarly publications that reported on specific aspects of Mach. The combination of working code and evangelism created enthusiastic support for the Mach vision in industry and academia. A consortium of companies (led by Digital Equipment Corporation and initially including Apollo Computer, Groupe Bull, Hewlett-Packard, IBM, Nixdorf Computer, and Siemens AG) formed the "Open Software Foundation (OSF)" and embraced Mach as the foundation of their operating system efforts. The USENIX Association organized multiple Mach-centered conferences in the early 1990s: the USENIX Mach Symposium at Monterey, CA in November 1991, the USENIX

Mach III Symposium at Santa Fe, NM in April 1993, and the USENIX Microkernels and Other Kernel Architectures Symposium in San Diego, CA in September 1993. An indication of Mach's mindshare is suggested by the title of the keynote talk at the USENIX Experiences with Distributed and Multiprocessor Systems symposium in San Diego, CA in September 1993. At this conference the keynote address by Professor Larry Peterson was titled, "Is There Life After Microkernels?" Mach concepts clearly dominated the discourse of the operating systems research community in the early 1990s.

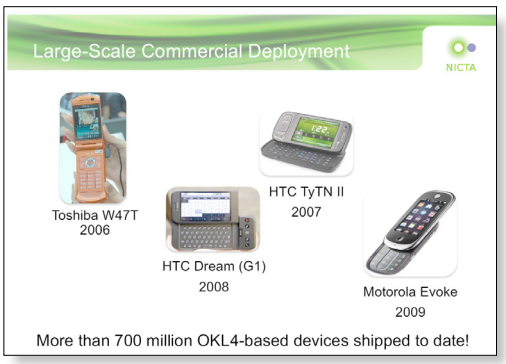


FIGURE 2: Commercial Use of L4 Microkernel (Source: Gernot Heiser)

Yet, by the end of the 1990s this momentum had dissipated. Digital Equipment was no longer an independent company, having been absorbed into Compaq and Intel. The Open Software Foundation was a spent force. The Mach microkernel no longer dominated efforts in commercial operating systems. It was Linux, with a traditional monolithic kernel structure, that was rapidly gaining a foothold in industry. There are ongoing debates about the cause of this turn of events, but establishing a specific causal chain is difficult. The Mach microkernel vision of simplicity and extensibility continued to inspire academic research, as evidenced by work such as SPIN [3] and Exokernel [5]. However, it was no longer seen as a competitive advantage in the marketplace.

Fast forward to 2012. It would seem inevitable that after a further 10–15 years, there would be little left of microkernels in terms of active influence on present-day operating systems. Fortunately, this is not true! The key person who sustained the microkernel vision after Mach's influence faded was Jochen Liedtke (1953-2001). His efforts were later amplified and extended by others such as Hermann Härtig and Gernot Heiser. In a series of papers in the mid-1990s [13, 14, 10], Liedtke and his associates explored microkernel architectures from first principles in the context of the L3 and L4 operating systems. They established that many of the performance challenges of microkernels could be overcome through careful design and attention to implementation details. Today, Heiser's team at the University of New South Wales and NICTA is the center of academic and commercial efforts surrounding the L4 microkernel [12]. Figure 2 (reproduced here by permission of Gernot Heiser) illustrates the extensive use of L4 in embedded systems.

3. Multiple Personalities

Mach's second big idea was the ability to support multiple operating system personalities on a common base. Figure 3 shows the earliest published reference that I could find for this concept. This new capability was especially attractive to companies that were being forced

processes. The reason this implementation strategy is so attractive for open systems, is that it can allow more than one operating system environment to be supported on the same machine, on the same kernel, at the same time. Systems such as UNIX or OS/2 could potentially co-exist in their native form. The kernel becomes a kind of universal "socket" into which more than one operating system environment can be plugged, insulating that software from the hardware itself and greatly simplifying its design and maintenance.

FIGURE 3: Earliest Mach Reference (1989) to the Concept of Multiple Personalities [17]

to make difficult choices between operating systems on their new hardware. For example, on its new PowerPC hardware platform IBM had to choose between supporting Microsoft's Windows 3.1, its own OS/2, or its AIX flavor of Unix. On its new Alpha hardware platform, Digital Equipment faced a similar decision between VMS and OSF/1. Supporting multiple operating systems on the same hardware was theoretically possible, but it would have been very expensive in terms of software development and maintenance. Mach's ability to support multiple personalities on a common base appeared to be a cost-effective approach to solving this problem.

In 1992, a development team from Digital Equipment described a proof-of-concept implementation of VMS on Mach 3.0 [20]. Figure 4, reproduced from that work, illustrates the layering of VMS on Mach. Their work provided independent confirmation that multiple OS personalities could be supported on the Mach microkernel. At the same time, it exposed certain limitations. For example, it proved impossible to accurately emulate VMS scheduling policies using Mach. As another example, it was not possible to emulate VMS' strong isolation of kernel resource usage by different users. Preliminary measurements also suggested that layering VMS on Mach resulted in unacceptable performance overhead. Due to these technical concerns and other nontechnical considerations, Digital Equipment did not follow through with a production quality implementation of VMS on Mach.

A much bigger bet was made by IBM. Rather than a tentative exploration, it wholeheartedly embraced the concept of multiple personalities on a common base. In 1991, IBM initiated a major development effort called Workplace OS that would unify its diverse menagerie of operating systems that included OS/2, PC DOS, AIX and OS/400. Fleisch et al [6, 7] and Rawson [18] give candid retrospective accounts of this effort. Figure 5 from Rawson's account illustrates the structure of the system. At its peak, over 2000 software developers spanning multiple IBM divisions were engaged in implementing Workplace OS. Starting from the Mach 3.0 kernel, there was initially rapid progress. OS/2, DOS, and Unix personalities on PowerPC and Intel hardware were demonstrated by IBM at COMDEX in Fall 1992. But a year later, in Fall 1993, IBM announced that AIX would remain a separate system outside Workplace OS. Version 1 of the IBM Microkernel was released in Fall 1995 and adopted

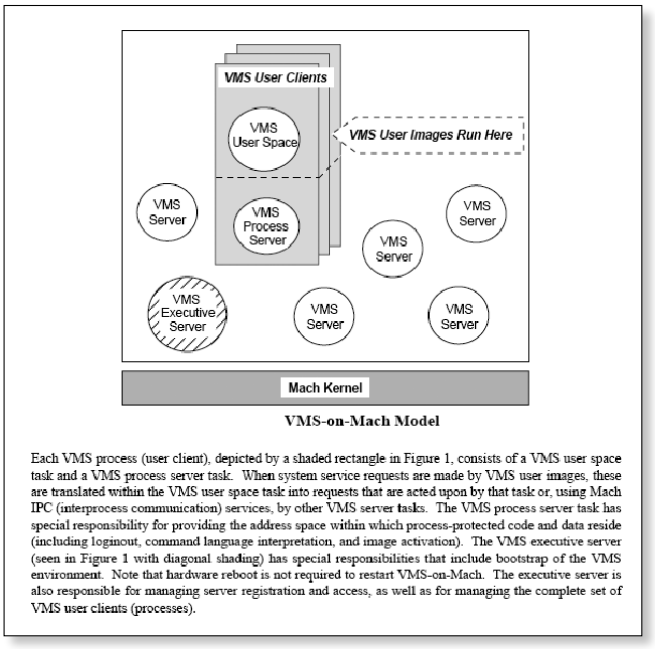


FIGURE 4: Digital Equipment's VMS on Mach Microkernel Prototype (Source: Wiecek et al [20])

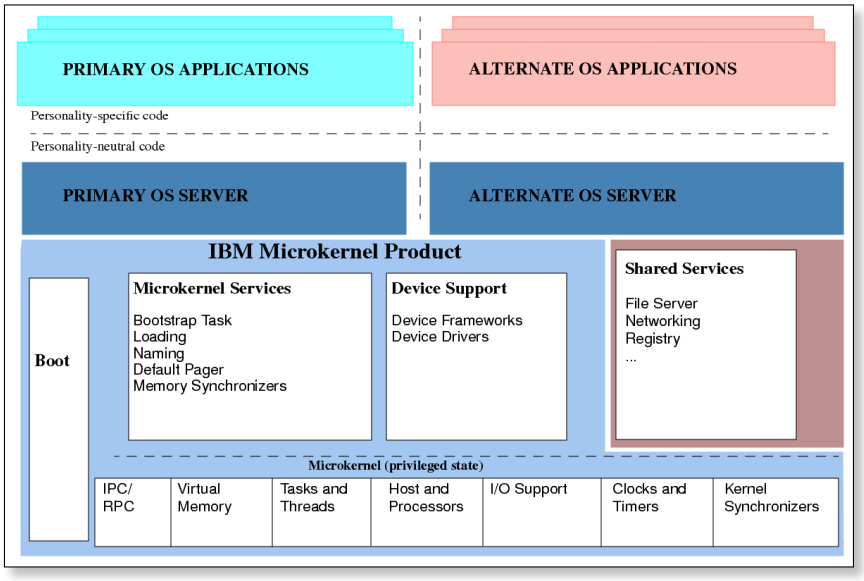


FIGURE 5: Structure of IBM's Workplace OS (Source: Rawson [18])

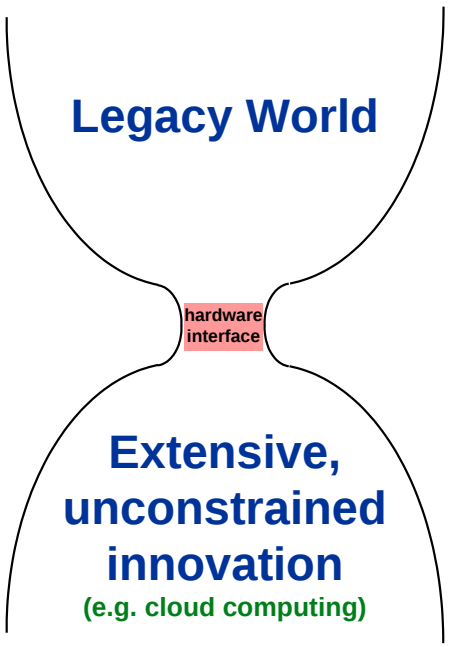


FIGURE 6: Legacy-compatible VM Ecosystem

by a few companies and universities. But a few months later, in late 1995, the entire Workplace OS project was cancelled.

As with the microkernel concept, the 1990s ended with the apparent repudiation of Mach's vision. Yet, a dozen years later, we see exactly this vision transformed into everyday reality: various flavors of Windows and Linux coexisting on a common base in cloud computing. Each of these diverse environments is encapsulated within a virtual machines (VM), and the "common base" is a virtual machine monitor (VMM) or hypervisor. It is interesting to speculate on why this approach has succeeded almost effortlessly, when the same end goal proved so hard to attain earlier on a different foundation.

VMs are an even older idea than microkernels or multiple personalities. They were not created for aesthetic or philosophical reasons, but as a very pragmatic solution to a real problem. VMs were invented by IBM in the mid-1960s as a timesharing approach that enabled concurrent, cleanly isolated system software development on a single mainframe by multiple programmers. Since mainframe hardware of that era was expensive, VMs were a cost-effective approach to enhancing programmer productivity by providing a private "mainframe" for each developer rather than reserving dedicated time on real hardware. Accuracy of hardware emulation was paramount because the system software developed on a VM was intended for use in a mainframe operating system. That software had to work with negligible changes on the real hardware. The techniques for efficient and accurate hardware virtualization that were developed in this era have proved to be of lasting value.

Figure 6 illustrates the value proposition of the VM abstraction today. A large legacy world of software, including operating system software, represents a substantial intellectual and financial investment that has been already been made on existing computer hardware. This legacy world can be completely closed source: there is no requirement for availability of source code, nor a requirement for recompilation or relinking. All that is needed is standard software installation, just as on real hardware. Beneath the implementation of the emulated hardware interface, there can be extensive innovation along many dimensions. This innovation is totally isolated from the idiosyncrasies of the legacy world above. Certain specific attributes of the VM abstraction accounts for its longevity and success. The hourglass shape in Figure 6 could depict the interface specification of any abstraction (thin waist), applications dependent on this interface (above), and implementations of this interface (below). More specifically,

it could represent an execution engine such as the Java Virtual Machine (JVM) [15] or the Dalvik Virtual Machine for the Android platform [21]. While these alternatives (collectively referred to as software virtualization) have successful niches, they do not approach the VM abstraction (also referred to as hardware virtualization) in terms of longevity, widespread usage and real-world impact. Why is hardware virtualization so much more successful? First, the interface presented by the VM abstraction is compatible with legacy operating systems and their valuable ecosystems of applications. The ability to sustain these ecosystems without code modifications is a powerful advantage of VMs. The ecosystems supported by software virtualization tend to be much smaller. For example, a JVM is only valuable in supporting applications written in specific languages such as Java. In contrast, a VM is language agnostic and OS agnostic. In fact, a JVM can be part of the ecosystem supported by a VM. Hardware virtualization can thus subsume software virtualization. Second, a VM interface is narrow and stable relative to typical software interfaces. In combination, these two attributes ensure adequate return on investments in the layer below. By definition, a narrow interface imposes less constraints on the layer below and thus provides greater freedom for innovation. The stability of a VM interface arises from the fact that the hardware it emulates itself evolves very slowly and almost always in an upward compatible manner. In contrast, the pliability of software results in more rapid evolution and obsolescence of interfaces. Keeping up with these changes requires high maintenance effort. Pliability also leads to widening of narrow interfaces over time, because it is difficult to resist the temptation to extend an interface for the benefit of a key application. Over time, the burden of sustaining a wide interface constrains innovation below the interface.

The importance of the narrowness and stability of an interface can be seen in the contrasting fortunes of process migration and VM migration, which are essentially the same concept applied at different levels of abstraction. Process migration is an operating system capability that allows a running process to be paused, relocated to another machine, and continued there. It has been a research focus of many experimental operating systems built in the past 20 years. Examples include Demos [16], V [19], Mach [23], Sprite [4], Charlotte [2], and Condor [22]. These independent validation efforts have shown beyond reasonable doubt that process migration can indeed be implemented with acceptable efficiency. Yet, in spite of its research popularity, no operating system in widespread use today (proprietary or open source) supports process migration as a standard facility. The reason for this paradox is that a typical implementation of process migration involves such a wide interface that it is easily rendered incompatible by a modest external change such as an operating system upgrade. Long-term maintenance of machines with support for process migration involves too much effort relative to the benefits it provides. The same concept of pausing an executing entity, relocating it, and resuming execution can be applied to an entire VM rather than a single process. This is VM migration, a capability that is in widespread use in cloud computing systems today. Guest OS changes do not affect an implementation of VM migration. This insulation from change, embodied in the narrow and stable VM interface, is crucial to the real-world success of VM migration.

The relationship between microkernels and VMMs has been the subject of debate in the research literature. In 2005, Hand et al published a position paper entitled, "Are Virtual Machine Monitors Microkernels Done Right?" [8]. In response, Heiser et al offered a rebuttal [11]. The debate continues!

4. Closing Thoughts

More than a quarter of a century has passed since Rick first envisioned Mach. Yet, looking at Figures 1 and 3, one is struck by how relevant those ideas are even today. The attributes of simplicity and extensibility continue to inspire operating system research, even as commercial operating systems grow ever more bloated in size and complexity. From a user's viewpoint, the ability to seamlessly and effortlessly switch between operating system worlds continues to be valuable. Mach pointed the way, and it remains the intellectual forerunner of so many aspects of today's systems.

References

[1] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and, Young, M. Mach: A New Kernel Foundation for UNIX Development. In *Proceedings of the Summer USENIX Conference* (1986), pp. 93–112.

[2] Artsy, Y., and Finkel, R. Designing a Process Migration Facility: The Charlotte Experience. *IEEE Computer 22*, 9 (1989).

[3] Bershad, B. N., Savage, S., Pardyak, P., Sirer, E. G., Fiuczynski, M. E., Becker, D., Chambers, C., and, Eggers, S. Extensibility Safety and Performance in the SPIN Operating System. In *Proceedings of the Fifteenth ACM symposium on Operating Systems Principles* (Copper Mountain, CO, 1995).

[4] Douglis, F., and Ousterhout, J. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software Practice and Experience 21*, 8 (1991).

[5] Engler, D. R., Kaashoek, M. F., and, O'toole, Jr., J. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, 1995).

[6] Fleisch, B. D. The Failure of Personalities to Generalize. In *Proceedings of the 6th IEEE Workshop on Hot Topics in Operating Systems* (Cape Cod, MA 1997).

[7] Fleisch, B. D., Co, M. A. A., and, Tan, C. Workplace Microkernel and OS: A Case Study. Tech. Rep. CS984, Department of Computer Science, University of California at Riverside, April 1998.

[8] Hand, S., Warfield, A., Fraser, K., Kotsovi Nos, E., and, Magenheimer, D. Are Virtual Machine Monitors Microkernels Done Right? In *Proceedings of the 10th conference on Hot Topics in Operating Systems* (Santa Fe, NM, 2005).

[9] Hansen, P. B. The Nucleus of a Multiprogramming System. *Communications of the ACM 13*, 4 (April 1970), 238–250.

[10] Hārtig, H., Hohmuth, M., Liedtke, J., Wolter, J., and, Schōnberg, S. The Performance of µkernel-based Systems. In *Proceedings of the sixteenth ACM symposium on Operating Systems Principles* (Saint Malo, France, 1997).

[11] Heiser, G., Uhlig, V., and, Levasseur, J. Are Virtual Machine Monitors Microkernels Done Right? *SIGOPS Operating Systems Review 40*, 1 (January2006).

[12] Klein, G., Elphinstone, K., Heiser, G., Andron Ick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and, Winwood, S. seL4: Formal Verification of an OS Kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles* (October 2009).

[13] Liedtke, J. Improving IPC by Kernel Design. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles* (Asheville, NC, 1993).

[14] Liedtke, J. On Microkernel Construction. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, 1995).

[15] Lindholm, T., and Yellin, F. The Java Virtual Machine Specification (2nd Edition). Prentice Hall, 1999.

[16] Powell, M., and Miller, B. Process Migration in DEMOS/MP. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles* (Bretton Woods, NH, Oct. 1983).

[17] Rashid, R., Baron, R., Forin, A., Golub, D., Jones, M., Julin, D., Orr, D., and Sanzi, R. Mach: A Foundation for Open Systems. In *Proceedings of the Second Workshop on Workstation Operating Systems* (1989).

[18] Rawson III, F. L. Experience With the Development of a Microkernel-Based, Multiserver Operating System. In *Proceedings of the 6th IEEE Workshop on Hot Topics in Operating Systems* (Cape Cod, MA, May 1997).

[19] Theimer, M., Lantz, K., and Cheriton, D. Pre-emptable Remote Execution Facilities for the VSystem. In *Proceedings of the 10th Symposium on Operating System Principles* (Orcas Island, WA, December 1985).

[20] Wiecek, C. A., Kaler, C. G., Fiorelli, S., William C. Davenport, J., and, Chen, R. C. A Model and Prototype of VMS Using the Mach 3.0 Kernel. In *Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architectures* (Seattle, WA, April 1992).

[21] Wikipedia. Dalvik (software). http://en.wikipedia.org/wiki/Dalvik_(software), September 2011.

[22] Zandy, V., Miller, B., and Livny, M. Process Hijacking. In *8th International Symposium on High Performance Distributed Computing* (Redondo Beach, CA, August 1999).

[23] Zayas, E. Attacking the Process Migration Bottleneck. In *Proceedings of the 11th ACM Symposium on Operating System Principles* (Austin, TX, November 1987).

A Letter

Alfred Spector

ick Rashid has had a direct influence on the lives of thousands, or perhaps tens of thousands of family, friends, and colleagues. And as one of my closest collaborators during our professorial days at Carnegie Mellon in the 80's, Rick certainly had a big influence on me. He had been at CMU one or two years when I arrived, and we formed a good team about a year after I joined. We collaborated on a wide collection of issues in the operation of the department and in computer science.

As I think back through the decades in which I have known Rick, what first comes to my mind is his genuine, heart-felt optimistic outlook: I think that both in general life and in technology, Rick believes things are likely to turn out well—at least as long as one pursues them with energy, creativity, and intelligence. For example, Rick used to be able to say something to the effect of, "bigger, better, faster, cheaper, …" more frequently and fluidly than anyone else. Partially, I recall the words flying off his tongue just because he was happy. But, mostly, I think he was conveying that everything always just gets better.

Cheerful optimism (followed by deep focus on making things happen) is a good attribute for a research leader and entrepreneur. And it's a particularly good one in the domain of computer science, where I believe almost anything and everything is achievable if attacked with great gusto and discipline. There are endless good ideas and vast numbers of ways to design and architect software systems. You just have to choose (focus!) and work like hell to make them a reality.

Coming to my mind second is Rick's incredible breadth of interests. There is hardly a topic on which he doesn't have an interest, or for that matter, an opinion. There are few people with his breadth of curiosity and engagement. This is a trait I and other research leaders share, but Rick is less likely than most to apply restrictive filters. I admit to thinking that some things he was interested in were not terribly pragmatic, but he's often been right. Who is now to say

that video games are not a pragmatic focus of computer scientists? (Now, whether they are a good use of the time of our youth...but, that's a different question and a different essay...)

In this quick note, I'd like to add that technically, Rick's and my ideas on systems were quite compatible and symbiotic. They profoundly influenced the domain of my work at CMU. As I write this, I am thinking I probably would have worked on low-level communication mechanisms if Rick had not already been working on them in Accent and in Mach. So, instead, my work moved to higher-level abstractions in the distributed computing space, work which could layer on his. So, Rick and I would frequently co-present to funding agencies, but we somehow did our presentations bottom-up with Rick focusing on processes, low-level communication, and local storage, and me later focusing on transactions, replication, etc. That implied that he would present his micro-kernel ideas (to which I would dutifully listen for the n+1st time), and then I would present my team's work. While I had to listen to his talks, this was not symmetric as he often had an excuse to depart before I spoke.

So, I admit that I got a bit sick and tired of listening to the microkernel story after a while, but I affirm this is not the main reason why I left CMU to found Transarc. It is, however, partially why—when Rick called me one evening at my 20th floor office in the Gulf tower asking me whether he should take Microsoft's job offer in 1991—I still recall definitively advising Rick to take the job and to try something new and broader. I felt he had so much more breadth, and that's proven true. So, all of you at Microsoft Research who have benefited from Rick's presence, you should know the present head of research at Google was at least a small part of the reason that you've had Rick as your leader the last few decades.

How I Came to Work for Mach's Fearless Leader

Jonathan Chew

Me

- Worked on Mach with/for Rick from 1986 to 1991
- Went to UC Berkeley
- Didn't know what to study until I stumbled across CS as a prerequisite to a Business major and never imagined where I would get to work
- Worked at PERQ on Accent OS (predecessor to Mach)
- Came to work for CMU CS dept. after PERQ went out of business

Rocky & Bullwinkle?

- Rocky & Bullwinkle Show was cartoon that aired on network television in US from 1959-1973
- Unofficial mascots of the Mach project
- Named some of our machines after characters from the show
- I think of some of the characters when I think of Rick

Bullwinkle J. Moose & Rocket (Rocky) J. Squirrel

Fearless Leader

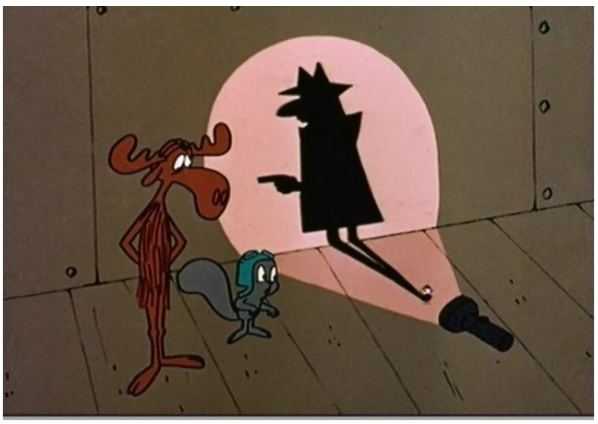
- Fearless Leader
  - Dictator of Pottsylvania
  - Boss of Boris and Natasha who were Rocky and Bullwinkle's adversaries
- Rick was our Fearless Leader
  - Coincidentally, we were in Pennsylvania which seems similar to Pottsylvania
  - Rick stuck up for our project and us

Boris Badenov, Natasha Fatale, and Fearless Leader

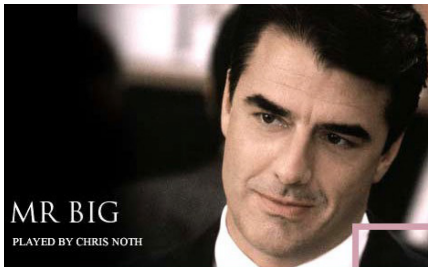
- Must have been a good salesman because CS dept ran Mach on most of their machines
- Great spokesperson because we got to meet and/or work with Bill Gates, Steve Jobs, BBN, IBM, Encore, Sequent

Mr. Big

- Boss of Fearless Leader
- Rick was Mr. Big for Mach
- Everyone working on Mach worked for Rick: Avie, Mike Young, Dave Golub, Bob Baron, Bill Bolosky, Mary Thompson, Sandro, Rich Draves, Rich Sanzi, David Black, Mike Jones, Joe Barrera, Dan Julin, Mark Stevenson, Randy Dean, me…



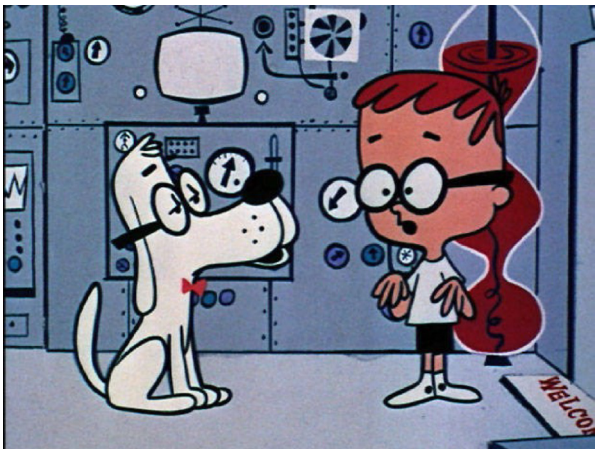
Bullwinkle, Rocky, and Mr.Big



- Could argue that the head of CS department or Howard Wactlar who was Vice Provost of Research Computing was Mr. Big though…

Mr. Peabody

- Genius dog with his boy Sherman who traveled through time using his WABAC (Wayback) machine
- Rick
    - Genius, wears glasses, and definitely had students who were bright, young, naïve, and enthusiastic like Sherman.
    - Has "Rashid Time" instead of a time machine.
    - Rashid Time is Rick's time estimate for a given task which can be converted to normal human time by doubling and going up to next unit of time (e.g., 1 Rashid week = 2 months for normal human beings)

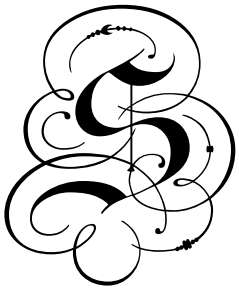


Mr. Peabody, Sherman, and WABAC machine

Conclusion

- Mr. Big and Fearless Leader were bad guys
- Rick is a good guy
    - At least our Mr. Big and Fearless Leader for Mach in name
    - Maybe most like Mr. Peabody
- I consider Rick to be a great leader, teacher, and friend to me.
- Will always think of "Rashid Time" whenever I think that something is hard or will take a long time.

Jonathan Chew and Family



tudents

Evolution of the Mach Operating System

From Research Project to Powering a Billion Devices

Avadis Tevanian, Jr.

I had the good fortune of joining Carnegie Mellon in the Fall of 1983 as a new graduate student in the computer science Ph.D. program. I was attracted to Carnegie Mellon by its focus on practical research. It was definitely a place to do research that was very tangible in nature as opposed to programs at other fine schools that seemed somewhat more theoretical.

I quickly bonded with Rick Rashid and he became my main advisor. Like me, Rick had spent time at the University of Rochester. I had just finished receiving a B.A. in Mathematics from Rochester. While it was a math degree, I had spent most of my time working with graduate students in their advanced computer science lab, which included a full lab of Xerox Altos. Rick had received his Ph.D. in Computer Science from Rochester prior to my arrival and so we had a natural connection. Rick was also one of the Principal Investigators in what was clearly a very practical research project—the Accent Operating System.

Accent was the operating system that powered the PERQ computer. This meant this research project had a direct influence on real world, commercial computer systems. While that was certainly incredibly exciting it felt a little bit too mature for me. After all, Accent was being fully deployed as part of a commercial product at PERQ systems. The team working on Accent was quite large, both inside and outside of the CMU CS Department, and it seemed as though it would be hard to do really interesting new work in the context of this mature project.

Luckily, as I was entering the Ph.D. program, there was the possibility of starting a new, related project. DARPA wanted to fund additional research in the operating system area with a focus on large multiprocessor systems. Digital Equipment Corporation (DEC) was in the process of designing such a system but it would need a new OS to power it. This work was certainly in Rick's area and when I mentioned to him that I really didn't want to work on Accent he suggested I start on this new project.

It was quite some time but we eventually named the project Mach. It took a while to get the name. We went through many ideas. One idea was MUCK (Multiprocessor User Communication Kernel). This idea came about on a rainy day as we walked to lunch in Oakland. The rain had left a number of puddles and we had to navigate a good amount of "muck." The name was clearly a joke, but when we returned to Wean Hall after lunch Rick jokingly mentioned it to one of our Italian colleagues (Dario Giuse) who misheard it as Mach. The name stuck and has served well for many years!

Technically speaking, we knew early on that we wanted to build a lot of new technology. But we also liked how Accent could be used by others completely unaffiliated with the Accent project itself. We wanted our work to have the potential to be widely used. At this point in time, Berkeley Unix distributions were very popular in the university setting so we elected to leverage that work.

We started our work before DEC even had a prototype hardware system for us to use. DEC was building on the VAX 780 system and was going to build a system dubbed the "784." The 784 would be four 780's interconnected with a shared memory subsystem. While waiting for the 784 to become real we did some early work on single processor 780 systems. DEC had also just released a new workstation, the MicroVAX, which had the same overall architecture of the 780 but in a workstation form factor.

Some of the earliest coding was done on these systems, but the first real parts of Mach where done during one of those summers. I was fond of spending my summers in Maine on Sebago Lake where my family was. I was able to take a MicroVAX with me for the summer and started coding the thread subsystem. It was a scary prospect. There was only one such system available to me. The system ran BSD, but if I got myself into an unbootable state there would be no way to recover! So each and every iteration was done very carefully. By the end of the summer I had the beginnings of the Mach kernel up and running—it was largely a BSD system with a new thread scheduler.

Back at CMU I eventually coded up the first version of the virtual memory system. The IPC system was largely lifted and evolved from Accent. I don't remember who integrated it in but it might have been Rick. The VM system was novel in that it allowed for very flexible functionality (some of which had been available in Accent) in a very portable way. Portability wasn't a priory important to us but we knew it was likely to pay off in the future.

The system was made to be scalable for multiprocessing and eventually we had a working VAX 784 in our lab. Our OS powered the 784 just wonderfully and we were finally able to get user processes to run in parallel! Our research attracted the attention of vendors who were pushing into multiprocessing and our portability paid off with ports to Sequent and Encore systems.

As researchers we of course published our work. At one of the USENIX conferences I presented one of our papers. This conference was attended by some of the software engineers that were working at NeXT Computer, although we were not aware of that at the time.

NeXT Computer was founded in 1986 shortly after Steve Jobs was kicked out of Apple as the result of a boardroom showdown with John Sculley. NeXT was building a completely new computer targeting the higher-education space. NeXT was very secretive so we knew very little of their plans.

In the fall of 1986 Rick and I headed to California to attend a BSD-related conference at UC Berkeley. Steve Jobs heard that we would be in his neighborhood and invited us to dinner. We stopped by the NeXT office in Palo Alto and met with Steve just prior to dinner. He told us a little bit about what NeXT was doing and then dropped a bombshell—his team had seen our presentation at USENIX and had decided to base their new computer on Mach.

Rick and I were certainly happy about this news but we could in no way foresee the importance of the decision NeXT had made. The possibility of our research making it into what would likely become a mainstream product motivated me to get my thesis work in gear. This allowed me to finish in what was probably regarded as near-record time for computer science Ph.D. students at CMU.

NeXT clearly had a leg up in getting me to sign on as an employee, but Microsoft, Sun and Apple all made attempts. The attraction of taking our research into a real commercial product was quite exciting and it didn't hurt to go to work with Steve Jobs. Bill Gates was also a strong pull to Microsoft. Bill was actually more interesting to talk to about technical topics, but alas it wasn't clear how my research would be relevant to Microsoft. NT work was already under way and there was even another group working on OS/2. There seemed to be some political infighting so NeXT was the right choice.

When I arrived at NeXT they of course had already started the porting work. NeXT's first computers were based on the 68000 architecture and they had prototypes already working when I arrived. What amazed me was the level of adoption the NeXT engineers had in the new technologies we had built. They were eager to dive right in and use all of the new features we had built. Of course, our decision to be BSD-compatible made this all possible as we had a huge set of software to leverage.

One of the first technologies to be widely adopted was threads. These were very new to the Unix environment at the time and proved to be extremely useful for building graphic user interfaces. Threads were key to implementing a very responsive user interface, allowing the offloading of non-time-critical functions to background threads and to help manage devices like the on board digital signal processor for audio. The first NeXT Computer had a new, unique mass storage device—an optical drive. The optical drive was unlike any other storage at the time. The OS couldn't simply write bits to this media. In order to put data on the media it had to erase, write and then verify the data. The write might fail and so the verify was important to check the bits. We had to build the software for this device from scratch, including bad block management. Threads were key to making this device work.

Virtually every other feature that Mach offered was quickly adopted by the NeXT software team. Threads were used throughout. NeXT's Objective-C language (a combination of Smalltalk and C) was extended to allow object messaging between processes (Distributed Objects).

This object messaging was implemented by integrating Objective-C's runtime with Mach's IPC mechanism. It specifically encouraged large data messages knowing that the underlying virtual memory system could handle the copies using copy-on-write. One particularly clever use of Distributed Objects (and hence, IPC) was the pasteboard server. The pasteboard server implemented copy and paste. Because of the copy-on-write nature of the underlying system a user could copy data, e.g., a huge image or document, in an instant and then paste it in an instant, all with no actual copying occurring.

Portability of Mach again proved to be very valuable. NeXT had no plans to use anything other than the 68000 line of processor but as time went on business needs changed. Over the next few years NeXT took on projects to use new processors in its own products as well as port to other computer systems. When the 68000 family started to run out of gas Motorola attempted to move to a new RISC architecture, the 88000. Although they never shipped, NeXT built prototypes based on the 88000. Porting was a breeze and made especially easy by the portability of the VM system. The 88000 used an inverted page table design which was easy to efficiently support. NeXT also ported Mach (and the entire software stack) to the Intel x86 architecture, Sun Sparc and HP PA-RISC. All ports were maintained as a single source, multi-architecture targeted project.

On December 20th of 1996 Apple announced that it would acquire NeXT. By this time I had given up my role as an engineer and moved along the management track. I started managing NeXT's OS team shortly after joining NeXT. By the time Apple decided to acquire NeXT I had become NeXT's VP of software engineering.

A key event leading to the acquisition of NeXT was a bake-off Apple staged between NEXTSTEP (NeXT's software) and BeOS (a competitor under consideration from Be, Inc.). During this bake-off Steve Jobs and I presented to Apple's senior management team. We were able to demo for them a full-featured OS that had capabilities that could be found nowhere else. We demonstrated complex media applications, DSP-enhanced science applications and even regular productivity applications all running together, seamlessly. At one point in the demo we had four QuickTime videos running simultaneously with animated science applications and other applications. This was enabled by all of the low-level functionality provided by the Mach kernel.

The acquisition closed in February of 1997 and I moved my office to Cupertino, along with my team. I was now in charge of all software engineering at Apple and found myself dealing with big company politics. Once inside Apple I found internal "antibodies" attempting to derail the entire purpose of the acquisition of NeXT. Perhaps it shouldn't have been a surprise given Apple's performance then but there were many managers and engineers with their own agendas and people just were not working together. While I had, perhaps naïvely, assumed that we would now just start working on a new OS for Apple based on NEXTSTEP I found instead that people inside of Apple wanted to revisit the OS roadmap.

Apple had no shortage of operating systems. It had its "classic" MacOS, then at version 7.5. It had also been working on Copeland. Copeland had been so badly managed that it had been

all but written off. Part of Copeland was a kernel called NuKernel. NuKernel was in many ways designed to be Mach-like and it was thought that NuKernel could host an updated Mac environment. There was Taligent, but no one gave it very serious consideration.

Apple already had MacOS 7.5 and many inside Apple thought we could use NuKernel, port some of the higher level NEXTSTEP technologies and also host some of the legacy Mac technologies. As an interesting aside, at this point in time Apple did not even have a TCP/IP networking stack. For TCP/IP they bundled a 3rd party add-on product called OpenTransport. In addition to giving consideration to NuKernel there were others that thought Apple should use Windows NT or Solaris. Both Microsoft and Sun were eager to see their operating systems adopted by Apple and fanning those flames.

While I was SVP of software engineering and responsible for delivering the new OS it wasn't clear I had the ability to make a decision about the operating system strategy. There were clearly a lot of politics involved, and besides, I was the new guy from a much smaller company.

Somehow, we just kept plugging away. We knew any new strategy would take some time to deliver so we decided to reinvest in the classic Mac OS. This led to a long string of hits (7.6, 8.0 and eventually 9.0) that served the existing customer base well. But we also knew the existing user base was small, and getting smaller. We needed the new operating system to appeal to non-Apple customers. With the support of Steve Jobs I was eventually able to get the company to commit to a NEXTSTEP-based strategy. Further, the political issues related to this strategy went away shortly after that summer. Gil Amelio (CEO) had been dismissed as had Ellen Hancock (CTO). Ellen was one of the biggest impediments to moving forward with a sensible strategy and with her gone it was essentially clear sailing. In retrospect it made no sense to use NuKernel. As I had argued at the time NuKernel still wasn't even done and we would no doubt encounter a large number of issues bringing it to market. Similarly, using NT or Solaris would have caused huge technical challenges in terms of porting lots of code that had already been designed to take advantage of Mach. But using NT or Solaris would have also had huge business impacts on Apple, including the loss of control of its OS.

With the roadmap issue dealt with the first task was to get NEXTSTEP software running on Mac hardware and to tweak the user interface to be comfortable for Apple customers. Porting to Mac hardware would seem to be an easy task, mostly around supporting PowerPC, except for many years Mac hardware teams had taken MacOS and modified it as they saw fit to adapt to their hardware. The version skew was worse than the PC world. Nevertheless, Mach was always very portable so the work was tedious, but not hard.

Around this point in time the OS team got very serious about making many parts of the kernel more dynamic in nature. This meant features like fully dynamic device drivers, file systems, network stacks, etc. These features all leveraged the basic Mach paradigms.

For users, battery performance was very important on laptops. As a result the kernel was made power-aware. Not only would it run efficiently on a laptop but it was enabled to do very fast sleep/wakeup (a feature that today is still unmatched) and eventually do a full deep sleep that requires no power consumption at all. The BSD networking stack was now a fully

integrated feature, but it was also possible to dynamically load other networking stacks. This was the way legacy support for Appletalk was done, for example. Networking was enhanced to work in a fully dynamic environment to support portables that frequently sleep/wake and even change locations.

With NEXTSTEP/Mach now running on a range of systems from laptops to desktops the next platform to support was Apple servers. This was a relatively straightforward effort as servers didn't require a lot of new functionality. However, something that became important, eventually, in the context of servers, was multiprocessing. In this regard Mach was ready for the task; after all, multiprocessing was an initial design point. Within just a short while of being acquired by Apple we now had Mach powering a range of Macintoshes from small laptops to desktops to big servers.

Mach portability continued to play an important role as NEXTSTEP/Mach morphed into Mac OS X. Through all those years of evolution, post acquisition by Apple, I insisted that we keep the software stack running on the x86 architecture. We had already ported NEXTSTEP to the x86 architecture earlier at NeXT, but at first blush Apple would have no need for such support. To me, it seemed inevitable that a time would come for Apple software to run on x86 hardware. It would either be in the form of a pure software product that ran on generic PCs or by way of Apple using x86 processors in its own products.

For many years, PowerPC and x86 fought a battle over performance, cost and power consumption. For a few years PowerPC had an advantage and that was good for Apple. But Motorola and IBM, the vendors for PowerPC, just couldn't keep up with Intel year after year, especially in areas important to Apple. Motorola could not keep up in performance on the desktop nor could they reliably deliver the yields Apple required on schedule. IBM had excellent PowerPC engineering but kept investing in the very high end. IBM was too expensive and their roadmap did not match Apple's roadmap, which required high volume, low cost, low-power processors for laptops and desktops. As expected, using x86 became inevitable.

When Apple decided to switch to x86 it was able to pull off an incredibly smooth transition. The low-level kernel did the heavy lifting of course, isolating all the differences in architecture. We had kept almost all of Mac OS X running on Intel processors for quite some time (in total secret). In fact, we had demonstrated it on occasion to PC vendors, Intel and others to try to drum up interest in the product—all to no avail. But now was going to be the payoff. A bunch of polish was required to bring up a handful of subsystems that hadn't been ported, mostly due to confidentiality issues (the port itself was as secret as any project at Apple). One key piece of software that was new was translation technology that allowed older PowerPC applications to be run on the x86 processor. This was licensed from a 3rd party and integrated into Mac OS X. Amazingly, old PowerPC-based applications could run completely transparently on Mac OS X on an x86 processor.

The transition from PowerPC to x86 for the Macintosh was technically an incredible feat. Business-wise it was as smooth as could be and could easily be the subject of future business analysis. It repositioned Apple as vendor of competitive hardware, able to focus on superior

software and industrial design. But the biggest technical transition for Mac OS X and Mach was still to come.

In the early days of Mac OS X we began experimenting with tablet designs. Microsoft and others had been working on tablets using pen/stylus input. We didn't think those products would ever gain traction—and they didn't. We experimented with touch-based tablets using multi-touch input. We prototyped many of the gestures that are commonplace now, e.g., pinch to zoom and others, and also prototyped touch-based keyboards. The OS worked fine on these devices, which at the time were based on PowerPC processors. All the prior work we had done to support laptops and mobility were very relevant. However, we didn't see a large market for such a product at the time, especially at the price points that would be required for that technology. The prototypes were put on the shelf.

At the same time Apple had found a hit on its hands—the iPod. iPods were flying off the shelves as everyone wanted them. At the time, the iPod software stack was a simple embedded stack designed specifically for the iPod. It had evolved over time along with the iPod, but it wasn't designed to really be expandable in any significant way and in fact was getting complicated due to a lack of general forward-looking design beyond just a simple iPod.

With the improvement of several technologies we realized we could build a very nice phone. We knew the iPod hardware could be enhanced sufficiently to be a phone and we had several software features that would lend themselves to a phone also (e.g., Dashboard widgets). Based on our prior work on the multi-touch tablet we knew we wanted our phone to be a keyboard-less multi-touch device. We determined that the iPod Touch and the iPhone could be very similar products. One would have the cellular radio and the other would not. But it would make sense to have the software be the same.

This led to a belief by some that the Apple phone and touch-based iPod should be based on a new software stack and that stack itself could be based on Linux and existing iPod software. Clearly, these devices would use ARM processors and Linux could already run on very low-end ARM processors. But I, and others, argued that the stack should be based on Mac OS X. In my view, Mach could be ported to the ARM processors that were likely to be used and if we did so we could reuse the entire software stack very easily. It was up to the Mac OS X team to prove that they could bring up the system on an iPhone prototype. Mac OS X was criticized for this application. Many assumed it wouldn't be able to boot fast enough, wouldn't be able to run in a small enough amount of memory, etc. But the bring-up team proved them wrong. In just a small number of weeks they had the core system up and running. As a bonus, the system was able to take advantage of all of the power-management and mobility features Apple had integrated into Mach/Mac OS X in the past, well ahead of anything done in the Linux space.

When Apple first shipped the iPhone it rebranded Mac OS X as OS X acknowledging that the OS scaled from these small, portable devices all the way up to servers. A bit later, in an effort to create some special mobile OS branding, Apple renamed the mobile OS to iOS. But iOS and OS X today are essentially the same. It's the same underlying Mach kernel.

The same application frameworks. Even applications have many similarities with the only real differences occurring at the final user interface. The core OS of both iOS and OS X are maintained by a single group today, all based on the same source code, with new features going back and forth between each "edition."

When I first starting working on what would eventually become the Mach project we did in fact target high-end, multiprocessor computing. But part of me knew we would build something fairly scalable. I envisioned lots of devices someday running the software we were about to build. After all, that's one of the reasons we architected it as a small kernel. Starting with that first MicroVAX used to bring up the threading model, Mach has evolved such that it is now, roughly 25 years later, the basis of devices ranging from the iPhone/iPod Touch, AppleTVs, laptops and desktops, all the way up to servers. In 2012 Apple will sell hundreds of millions of such devices.

And even today, while the Mach that exists to power OS X and iOS has undergone many revisions, much of the code we wrote back in the 1980's at CMU is still in there, under the hood.

A Letter

Michael Young

ick was one of my graduate advisors, from whom I learned a great many things in addition to systems-building skills. Some particular ones that I remember:

- Make it "bigger, better, faster, more efficient." Well, maybe not always bigger. In fact, making interfaces simpler was one of the more important systems lessons I learned from Rick. But the speed with which he rattled off this phrase still haunts me.
- Do not name your systems project MOOSE if you want your peers to take it seriously—name it for some boring physicist instead. Thankfully, this rule is not enforced at my current employer, Google.
- Things always run better on Rick's machine. That, and every project needs a David Golub.

One thing I didn't expect to learn from Rick was writing, but it may be the most important thing he taught me. Getting the writing done was one of the hardest parts of graduate school. Rick related a colorful analogy from his childhood: to generate a lot of <prose>, you need to sit down at a regular time every day and produce some. And so I did. What I produced first was a lot of technical detail, poorly organized. Rick helped me to outline my thoughts, and work them into a compelling story.

I haven't kept up with Rick much since leaving graduate school, but he has had a lasting influence. Thank you, Rick, for making my time at CMU both fun and educational.

A Letter

David L. Black

March 2, 2012
It is my pleasure to contribute this letter and a historic DH-CHAP Internet-Draft to the Festschrift for Rick Rashid.

earned my Ph.D. in computer science at Carnegie Mellon University in the latter part of the 1980s, working on the Mach operating system research project under Rick's supervision during much of that time. That was a very important time in my life, as I entered graduate school as a theoretician with an interest in systems and left as a serious systems builder based primarily on the work that I did on Mach. I learned an enormous amount from working with Rick that made the rest of my career possible.

It is ironic that I wound up working on an operating system research project, as I considered operating systems to be among the weakest computer science courses that I took as an undergraduate. That irony continued, as input/output was not a research focus of the Mach project—rather, it was an area of the operating system that had to be kept working in order to support interesting research elsewhere in the operating system. Despite that experience, in 1998 I joined EMC, a company whose technology (storage) was entirely focused on input/output, and within a year, I started working on what became the iSCSI storage networking protocol.

At that time, block storage networks for open systems (i.e., servers other than mainframes) were based on Fibre Channel Storage Area Networks (FC SANs). A block storage network provides server or host access to virtual or logical disks, in contrast to access to files and filesystems. FC SANs provide this access via support for SCSI; SCSI stands for Small Computer System Interface, but SCSI is used with computer systems of all sizes. iSCSI was based on the observation that Ethernet-based network switches had become comparable to Fibre Channel directors, and hence SANs could use Ethernet if only SCSI ran over TCP/IP.

iSCSI provided the functionality required to fill that gap, and iSCSI became the standard protocol for SCSI over TCP/IP. I was one of the designers of iSCSI, and a leader of iSCSI standardization as one of the chairs of the IP storage (ips) working group in the IETF (Internet

Engineering Task Force). The initial iSCSI standard was completed in 2003 and published as RFC 3720 in 2004. (An IETF standard is called an RFC, Request for Comments, for historical reasons.) iSCSI took off from there, and it is now a widely-deployed technology; according to IDC, revenue from iSCSI storage sales in 2011 exceeded $3 billion. Being involved in inventing the future is a thrill, especially when it works out in this sort of fashion!

I am now one of the chairs of the STORage Maintenance (storm) working group in the IETF, where the next version of the iSCSI standard is nearing completion. That standard is expected to be published as an IETF RFC later in 2012 or sometime in early 2013. I am a co-author of that forthcoming RFC, and it's my pleasure to dedicate that co-authorship to Rick Rashid in recognition of all that he did to make my career possible.

I am also contributing a historic IETF Internet-Draft on DH-CHAP (Diffie-Hellman Enhanced CHAP) as originally designed for iSCSI, although this authentication mechanism was not included in iSCSI. During the standardization of iSCSI, difficulties were encountered in selecting the mandatory-to-implement authentication mechanism. The first preference was a secure password protocol, SRP, but that became problematic due to patent concerns affecting the entire class of secure password protocols at the time. The primary alternative was CHAP (Challenge Handshake Authentication Protocol), but that mechanism had known security weaknesses, including a vulnerability to offline dictionary attacks mounted by a passive attacker who observed network traffic. DH-CHAP was designed as an alternative that avoided the patent concerns and improved on CHAP by preventing passive attackers from mounting offline dictionary attacks—an active attack (e.g., failed authentication attempt) is required in order to mount an offline dictionary attack. In contrast, secure password protocols effectively block all offline dictionary attacks.

A significantly strengthened version of CHAP was selected as the mandatory-to-implement authentication mechanism for iSCSI, and as a result, the DH-CHAP draft was never published as an IETF RFC. Subsequent Fibre Channel standardization included DH-CHAP as an authentication mechanism, although CHAP was also selected as the mandatory-to-implement mechanism for FC. The contributed DH-CHAP draft is in its last form (-01 version) before DH-CHAP work for iSCSI was abandoned in 2002. Although the draft is not 100% complete (e.g., Section 9 is entitled Open Issues), the draft contains a significant amount of useful information about the DH-CHAP design and its security aspects that complements the specification of DH-CHAP in the Fibre Channel security standards (FC-SP, Fibre Channel—Security Protocols, and the forthcoming FC-SP-2 standard).

Sincerely,

David L. Black, Ph.D.
Distinguished Engineer

Internet Draft
Document: draft-black-ips-iscsi-dhchap-01.txt
Expires: October 2002

D. Black
EMC

April 2002

DH-CHAP: Diffie-Hellman Enhanced CHAP for iSCSI

Status of this Memo

Abstract

This draft describes an authentication mechanism based on enhancing CHAP [RFC 1994] with a Diffie-Hellman Exchange (see Section 22.1 of [Schneier]) in order to prevent a passive eavesdropper from acquiring sufficient information to perform an off-line dictionary attack on the CHAP secret. The use of this authentication mechanism with iSCSI [iSCSI] is discussed, along with a brief comparison to the existing CHAP and SRP authentication mechanisms in iSCSI.

Caution should be exercised in drawing inferences from the fact that the author of this draft is one of the chairs of the IP Storage Working Group. This draft is an individual submission that the IP Storage Working Group is free to adopt, modify, reject, fold, spindle, and/or mutilate as it sees fit.

Table of Contents

1. Overview.....2
2. Conventions used in this document.....3
3. Basic CHAP Protocol.....3
4. Basic Anonymous Diffie-Hellman Key Exchange.....4
5. DH-CHAP, Diffie-Hellman Enhanced CHAP.....6
5.1 Bi-directional DH-CHAP.....7
5.2 DH-CHAP additions to DH and CHAP.....7
5.3 iSCSI text keys and values.....8
6. Brief Security Comparison of DH-CHAP with CHAP and SRP.....8
6.1 Passive Eavesdropping.....9
6.2 Use of Secret Information to Verify Authentication.....9
6.3 Impersonation and Man-in-the-Middle Attacks.....9
6.4 Reflection Attacks.....10
6.5 IPsec and DH-CHAP.....11
6.6 Passwords vs. Machine-generated keys.....11
7. External Authentication Server Considerations.....12
7.1 DH-CHAP and EAP.....12
7.1.1 Successful Authentication.....13
7.2.2 Unsuccessful Authentication.....13
8. Security Considerations.....13
9. Open Issues.....13
10. Change Log.....14
10.1 -00 to -01.....14
References.....14

1. Overview

DH-CHAP is a new combination of an unauthenticated Diffie-Hellman (DH) key exchange (see Section 22.1 of [Schneier]) with the existing CHAP algorithm [RFC 1994]. CHAP is vulnerable to an offline dictionary attack in that an eavesdropper who observes the CHAP challenge and response obtains information sufficient to test an unlimited number of candidates for the CHAP secret off-line. This offline attack succeeds more often than random chance would predict because CHAP secrets are often human-selected passwords, and humans aren't very good at selecting random passwords. For example, they often use words that can be found in a dictionary. DH-CHAP strengths CHAP in a fashion that requires an attacker to perform an online attack in order to capture the information required to mount an off-line dictionary attack. The three primary design goals of DH-CHAP are:

(1) Prevent a passive dictionary attack on CHAP via use of a DH exchange. An active attacker (e.g., impersonator or man-in-the-middle) can still gain sufficient information to mount an off-line dictionary attack.

(2) Stay as close to CHAP as possible. The ability to use existing RADIUS servers to verify authentication of DH-CHAP is desirable, although there are security considerations involved in doing so.
(3) Invent as little as possible. Every innovation in this sort of security protocol is an opportunity for subtle errors that can have major security consequences.

The basic idea behind DH-CHAP is to augment the CHAP challenge with the key that results from the Diffie-Hellman exchange so that the CHAP response depends on both of them. Section 3 describes the basic CHAP algorithm, Section 4 describes a basic unauthenticated Diffie-Hellman key exchange, and Section 5 specifies how DH-CHAP combines them.

2. Conventions used in this document

I - Initiator
R - Responder
| - Concatenation operation
H[] - One-way hash function

CHAP requires the MD5 one-way hash function for interoperability, and there is existing equipment that only supports MD5. SHA-1 is cryptographically preferable for new protocols. In order to use SHA-1, it will need to be registered in the IANA PPP Authentication Algorithms registry (http://www.iana.org/assignments/ppp-numbers, PPP AUTHENTICATION ALGORITHMS section).

Additional notation for each protocol is introduced in the section describing that protocol.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

3. Basic CHAP Protocol

CHAP is specified in [RFC 1994]. This is an authentication algorithm where the Initiator proves to the Responder that the Initiator knows a secret (e.g., password) that is also known to the responder without passing the secret in the clear to the Responder. The following notation is used here:

C_k - CHAP secret, shared by I and R
C_i - CHAP identifier, 1 octet
C_c - CHAP challenge, 16 octets when MD5 is used, MUST be generated from a source of real randomness
C_r - CHAP response, 16 octets when MD5 is used, computed from the above three

The CHAP secret, C_k is often a human-selected password, and hence is the source of CHAP's vulnerability to dictionary attacks. The CHAP protocol consists of the following steps:

(1) The Initiator requests service from the Responder and sends a list of acceptable CHAP hash algorithms:

I ---- algorithm list ----> R

(2) The responder selects a hash algorithm, generates a random challenge, C_c, chooses an identifier, C_i and replies:

I <---- algorithm, C_i, C_c ---- R

(3) The Initiator computes the response, C_r = H[C_i|C_k|C_c] and replies:

I ---- C_r ----> R

(4) The responder independently computes C_r' = H[C_i|C_k|C_c]. Authentication succeeds if C_r == C_r'. I and R both know C_i and C_c, hence this equality demonstrates to R that I knows C_k.

The computation of C_r' and the comparison may be outsourced to a RADIUS server or other external server capable of verifying CHAP authentication. R sends the username, C_i, C_c, and C_r to the server and asks if the response is correct. This allows the secret, C_k to be stored in the external server rather than the Responder.

RFC 1994 specifies that the CHAP Identifier (C_i) is returned at step (3). iSCSI prohibits this, and C_i is omitted from step 3 in the above description to match the iSCSI specification.

4. Basic Anonymous Diffie-Hellman Key Exchange

A description of the anonymous Diffie-Hellman key exchange protocol can be found in Section 22.1 of [Schneier]. This is a key exchange protocol where the Initiator and Responder agree on a secret key whose contents are computationally intractable to determine from the messages they exchange. For brevity, the acronym DH will be used to refer to this protocol. The following notation is used here:

- x - first random number
- y - second random number
- n - field prime
- g - generator

k - generated key

The random numbers MUST be generated from sources of real randomness. DH is based on modular (mod n) arithmetic in a finite field, which is often referred to as a group. The security properties of DH depend strongly on the size of the field and numerical properties of both the field prime and generator; see [Schneier] for further discussion. Current practice is to specify fixed values that are known to be secure for a small selection of different field sizes.

The protocol is shown here with Bob as the Initiator and Alice as the Responder in order to match the structure in which it is used by DH-CHAP.

(1) The Initiator requests service from the Responder:

I ---- request for service ----> R

(2) The Responder generates the first random number, x, computes g^x mod n and replies:

I <---- g^x mod n ----> R

(3) The Initiator generates the second random number, y, computes g^y mod n and replies:

I ---- g^y mod n ----> R

(4) Both parties can now calculate the resulting key:
- The Initiator generated y and received g^x mod n, and so calculates k = (g^x mod n)^y mod n = g^xy mod n .
- The Responder generated x and received g^y mod n, and so calculates k' = (g^y mod n)^x mod n = g^xy mod n .

A passive eavesdropper will find it computationally daunting to calculate g^xy mod n from the g^x mod n and g^y mod n that she can observe. In essence this requires calculating x from g^x mod n or y from g^y mod n, and calculation of these sort of discrete logarithms in a finite field is very difficult.

The key exchange ends here, but two more things usually happen when the exchanged key is used in practice:
(1) k and k' are employed in a fashion where something goes seriously wrong if k != k'.
(2) k and k' tend to be large numbers (e.g., kilobits) with low relative entropy, in that they possess randomness equivalent to true random numbers with far fewer bits. A one-way hash is a

useful tool to produce smaller numbers with higher entropy (e.g., that make good session keys).

5. DH-CHAP, Diffie-Hellman Enhanced CHAP

DH-CHAP is specified here. This is an authentication algorithm where the Initiator proves to the Responder that the Initiator knows a secret (e.g., password) that is also known to the responder without passing the secret in the clear to the Responder, or passing information between the parties that is sufficient for a passive eavesdropper to mount a dictionary attack against the secret.

The basic idea is to augment the CHAP challenge, C_c, with the key, k, that results from the Diffie-Hellman exchange so that the CHAP response, C_r, depends on both of them. The following additional notation is used here:

C_ca - the augmented challenge from which the DH-CHAP response is generated.

The DH-CHAP protocol proceeds as follows:

(1) The Initiator requests service from the Responder and sends a list of acceptable CHAP hash algorithms:

```
I ---- algorithm list      ----> R
```

(2) The Responder selects a hash algorithm, generates the initial CHAP challenge, C_c, and the first random number x. The Responder then computes g^x mod n, chooses a CHAP identifier, C_i, and replies:

```
I <---- algorithm, g^x mod n, C_i, C_c ---- R
```

(3) The Initiator generates the second random number, y, computes:
g^y mod n
C_ca = H[C_c|(g^x mod n)^y mod n] = H[C_c|g^xy mod n]
C_r = H[C_i|C_k|C_ca]

and replies:

```
I ----      C_r, g^y mod n ----> R
```

(4) The Responder now computes:
C_ca' = H[C_c|(g^y mod n)^x mod n] = H[C_c|g^xy mod n]
C_r' = H[C_i|C_k|C_ca']

Authentication succeeds if C_r == C_r'.

The computation of C_r' and the comparison could be outsourced to a RADIUS server or other external server capable of verifying CHAP authentication. R sends the username, C_i, C_ca', and C_r to the external server and asks if the response is correct. This allows the secret, C_k, to be stored in the external server rather than the Responder. A one-way hash is used to generate C_ca in order to make this possible (if C_ca were just C_c|g^xy mod n without the hash, it would be much larger than these servers expect, and using a subset of g^xy mod n's bits would sacrifice much of the strength of the key exchange). Outsourcing this verification has security implications that are discussed in Section 7.

5.1 Bi-directional DH-CHAP

Section 10.5 of [iSCSI] describes the use of two overlapped instances of CHAP to accomplish bi-directional authentication. (NOTE: This is not "mutual" authentication as the authentications in each direction are not cryptographically linked.) Each instance requires a hash on each side, so two instances would require two hash calculations (and in each case, one could be outsourced to a RADIUS server).

DH-CHAP adds two exponentiations and an additional hash calculation to each side. A simpleminded application of DH-CHAP to the two CHAP instances in the previous paragraph would result in a total of four exponentiations on each side to perform two Diffie-Hellman exchanges. Exponentiations are sufficiently expensive calculations to merit optimizing, and the optimization is straightforward. The g^xy mod n generated from a single DH exchange can be used for both DH-CHAP exchanges because in each case, g^xy mod n is concatenated to a different random challenge (C_c) before applying the hash that calculates C_ca and C_ca'. DH-CHAP may qualify as mutual authentication, because both participants "sign" the same key derived from a DH exchange, but mutual authentication was not among DH-CHAP's design goals.

5.2 DH-CHAP additions to DH and CHAP

The only addition that DH-CHAP makes to the base DH and CHAP protocols is the calculation of C_ca and C_ca' as a hash of the DH key appended to the transmitted CHAP challenge, specifically:

```
C_ca = H[C_c|(g^x mod n)^y mod n] = H[C_c|g^xy mod n]
C_ca' = H[C_c|(g^y mod n)^x mod n] = H[C_c|g^xy mod n]
```

This was added in this form to ensure that DH-CHAP can produce challenges and responses in the forms expected by existing servers that verify CHAP responses (e.g., RADIUS servers). All other

computations specified above and quantities sent in DH-CHAP messages are present in the original DH or CHAP algorithms.

DH_CHAP retains the challenge, C_c, for similarity to CHAP, and because it avoids having to invent a way to use the same DH exchange for both directions of bi-directional authentication. The existence of C_c poses a slight increase of difficulty to an attacker who has somehow defeated the DH exchange, but in practice defeating the DH exchange is primary obstacle facing such an attacker.

5.3 iSCSI text keys and values

DH-CHAP is a different algorithm from CHAP. To negotiate DH-CHAP for iSCSI authentication, a "DHCHAP" value needs to be added to the set of values defined for the AuthMethod key. The following keys need to be defined for DHCHAP's usage:

```
DHCHAP_A - DH-CHAP algorithm list or algorithm, similar to
      CHAP_A
DHCHAP_I - DH-CHAP identifier, similar to CHAP_I
DHCHAP_C - DH-CHAP challenge, similar to CHAP_C
DHCHAP_N - DH-CHAP name, similar to CHAP_N
DHCHAP_R - DH-CHAP response, similar to CHAP_R
DHCHAP_GX - g^x mod n sent from Responder to Initiator
DHCHAP_GY - g^y mod n sent from Initiator to Responder
```

When the DH-CHAP authentication method is the result of AuthMethod negotiation, the CHAP keys (CHAP_*) MUST NOT be used. When the CHAP authentication method is the result of AuthMethod negotiation, the DH-CHAP keys (DHCHAP_*) MUST NOT be used.

In addition, the DH_CHAP field prime and generator MUST be communicated from Initiator to Responder at step (1) (same approach as used for SRP with iSCSI):

```
DHCHAP_DHN - DH field prime, n in Sections 4 and 5
DHCHAP_DHG - DH generator, g in Sections 4 and 5
```

As is the case for SRP authentication in iSCSI, these are announced values that cannot be negotiated; the Responder MUST either accept them or close the connection. There is an open issue in this area; see Section 9.

6. Brief Security Comparison of DH-CHAP with CHAP and SRP

This section attempts to summarize important points of comparison among DH-CHAP and the CHAP and SRP authentication methods already specified in iSCSI. This is not intended to be a complete list,

but rather a listing of the important points. The author intends to revise this section to reflect important points from discussion on the mailing list as they develop.

6.1 Passive Eavesdropping.

CHAP allows a passive eavesdropper to gain information sufficient to mount an off-line dictionary attack on the CHAP secret. DH-CHAP prevents this, but in a fashion that may restrict DH-CHAP usage of existing servers (e.g., RADIUS) that verify CHAP authentication; see Section 7. SRP protects its secret from passive eavesdropping.

6.2 Use of Secret Information to Verify Authentication

SRP does not require secret information to be stored at the Responder if bi-directional authentication is not required; the SRP verifier can be made public without revealing secret information. Both CHAP and DH-CHAP require that the Responder or the system that verifies authentication for the Responder have access to the CHAP or DH-CHAP secret, C_k.

6.3 Impersonation and Man-in-the-Middle Attacks

An impersonation attack involves an attacker who does not know the secret impersonating a communicating party. A man-in-the-middle attack involves the attacker inserting himself between the communicating parties to read and modify communication between them. SRP prevents impersonation and man-in-the middle attacks from obtaining the secret or information sufficient to mount a dictionary attack on it.

Impersonation and man-in-the-middle attacks on CHAP and DH-CHAP disclose sufficient information to mount an off-line dictionary attack against the secret (C_k). CHAP and DH-CHAP Initiators are vulnerable to both sorts of attacks. CHAP and DH-CHAP Responders are not vulnerable to impersonation attacks because bi-directional CHAP and DH-CHAP as used in iSCSI require the Initiator to authenticate first, and forbid the Responder from sending its response if that authentication fails (see Section 6.4). CHAP Responders are vulnerable to man-in-the-middle attacks because the man-in-the-middle need only imitate a passive eavesdropper to succeed. DH-CHAP Responders are not vulnerable to man-in-the-middle attacks because the man-in-the-middle (M) has to conduct two independent Diffie-Hellman exchanges with I and R in order to capture the information required to mount a dictionary attack on DH-CHAP. In this case:

I <-- DH-CHAP <1> --> M <-- DH-CHAP <2> --> R

k (at I) and k' (at R) will not be the same because they depend on different random numbers whose generation (at I and R) M cannot control. As a result, authentication of I will fail because I will have sent a response (C_r) based on k which is different from k'; thus failure causes R to not send its response. The failed authentication of I occurs too late to protect I; M already has I's response, but this failure may serve as a warning that an attack may have taken place.

Impersonation attacks against one-way authentication are hard to detect because the impersonating attacker is not required to authenticate, and hence can simulate any one of a number of plausible reasons to terminate the connection after obtaining the information of interest. For SRP, this is of no consequence, as the attacker obtains no useful information about the secret, but this is a risk for both CHAP and DH-CHAP.

Impersonation attacks against bidirectional authentication may simulate failure of Initiator authentication (e.g., by closing the connection instead of responding). iSCSI Initiators MUST treat any login failure that causes bi-directional CHAP or DH-CHAP to fail to complete after the Initiator has sent its response as a potential security issue (e.g., treat the error in the same fashion as an authentication failure).

6.4 Reflection Attacks

The topic of reflection attacks was raised on the list, described as "the Target sends you a challenge, the Initiator sends the same challenge back to the Target". If the same CHAP or DH-CHAP secret (C_k) is being used for both directions of a bi-directional authentication, the Initiator and Target responses (C_r) are identical if the identifier (C_i) is also reflected. In this situation, if the Target responds to the challenge, it provides a rogue Initiator with the exact response (C_r) that is required to authenticate the Initiator to the Target. Needless to say, this must not be permitted to occur.

As CHAP and DH-CHAP are used in iSCSI, this reflection attack is almost prevented by the requirement that a Target MUST NOT continue if authentication of the Initiator fails. That requirement needs to be strengthened to require that a Target MUST NOT send its CHAP or DH-CHAP response if the Initiator has not successfully authenticated.

For example, the following exchange:

```
I->R     CHAP_A(A1,A2,...)
R->I     CHAP_A, CHAP_C, CHAP_I
I->R     CHAP_A, CHAP_C, CHAP_I
```

MUST result in the Responder (Target) closing the iSCSI TCP connection because the Initiator has failed to authenticate (there is no CHAP_R in the third message). In addition Initiators MUST NOT reuse the CHAP or DH-CHAP challenge sent by the Target for the other direction of a bi-directional authentication. Targets MUST check for this condition and close the iSCSI TCP connection if it occurs.

6.5 IPsec and DH-CHAP

Neither CHAP nor DH-CHAP defend against all active attacks such as impersonation and man-in-the-middle, and CHAP does not defend against a passive eavesdropper. In environments where these or other active attacks are a concern, DH-CHAP SHOULD NOT be used without additional protection. IPsec (IKE and ESP) provides the iSCSI defenses against these classes of attacks.

The iSCSI requirement for IPsec is "MUST implement," not "MUST use," and one can expect that administrators will choose not to use IPsec for a variety of reasons. To deal with such situations, the "MUST implement" iSCSI authentication mechanism needs to have an appropriate level of security on its own, although this level of security need not defend against all the attacks that IPsec prevents. For example, sending passwords in the clear and relying on IPsec to address all security issues would not be acceptable.

When IPsec is not in use, an attacker may choose to wait until authentication is complete and attack (e.g., hijack) the TCP connection, but attacking CHAP or DH-CHAP may yield something more valuable - a secret that could be used to authenticate in the future. Hence defending against dictionary attacks can still be important when IPsec is not in use.

6.6 Passwords vs. Machine-generated keys

The dictionary attacks against CHAP and DH-CHAP are based on the assumption that the CHAP and DH-CHAP secrets are human-generated passwords. If these secrets were instead machine-generated keys with sufficient randomness (e.g., 128 bits would suffice), vulnerability to dictionary attack would no longer be a concern (e.g., an off-line exhaustive search attack against a 128-bit CHAP key generated from real randomness is prohibitively expensive to mount). The downside of such keys is that they are difficult for

humans to handle and hence require administrative mechanisms to transfer and install keys (e.g., instead of writing the password on a scrap of paper, an administrator could carry a floppy between systems assuming that all systems involved have floppy drives). Systems that use IP Storage (especially iSCSI) may have a circular dependency if the IP Storage may be required to boot the system to the point that the mechanism to accept the key required to access the IP Storage becomes operational.

7. External Authentication Server Considerations

If a RADIUS or some other external server is used to verify DH_CHAP responses, the connection between the Responder and that server may be the weak link because the C_ca' and C_r sent over that connection provide sufficient information to mount a passive dictionary attack on C_k. If an eavesdropper can observe these values by monitoring that connection, DH-CHAP's additional protection against passive attack gained from the Diffie-Hellman exchange is lost. Any such connection to an external server to verify DH-CHAP responses MUST use confidentiality (e.g., IPsec ESP) or be protected from eavesdroppers via other means. Examples of "other means" include use of a separate isolated network for all RADIUS traffic to protect against eavesdroppers, and the use of traffic filters to prevent RADIUS traffic from escaping into areas of the network that are vulnerable to eavesdroppers. For bi-directional usage of DH-CHAP, this requirement also applies to any connection from an Initiator to an external response verification server.

7.1 DH-CHAP and EAP

The Extensible Authentication Protocol (EAP) (defined in [RFC 2284], being updated in [2284bis]) describes a framework that allows the use of multiple authentication mechanisms. This section (based loosely on section 6 of [EAP-SRP]) shows examples on how DH-CHAP might be used with EAP, but does not give the formal definition that would be needed to actually do so.

With the extensions to RADIUS (as defined in [RFC 2869]), the authenticating endpoint can be moved to a RADIUS server, or even beyond to a back end authentication server. This avoids some of the security issues discussed in the previous section on RADIUS by not exposing any more information that what is already exposed between the peer systems. Note that a RADIUS server would have to be upgraded though (in some way) to support EAP DH-CHAP, or any new EAP protocol.

In the following examples, the named parameters (g, n, C_i, C_c, C_r) are the same as described in the previous sections. DH-CHAP

represents the EAP Type that would be assigned for EAP DH-CHAP. The id parameter is used to match EAP-Responses with EAP-Requests. Note that if DH-CHAP is always used with EAP, the C_i parameter could be removed.

7.1.1 Successful Authentication

In the case where the EAP DH-CHAP authentication is successful, the conversation may appear as follows:

```
Authenticatee                  Authenticator
------------------------------ ------------------------------------
                               <- EAP-Request id=65 / Identity
EAP-Response id=65 / Identity
("Initiator")   ->
                               <- EAP-Request id=66 / DH-CHAP
                                  (g^x mod n, g, n, C_i, C_c)
EAP-Response id=66 / DH-CHAP
(g^y mod n, C_r)           ->
                               <- EAP-Success id=67
```

7.2.2 Unsuccessful Authentication

In the case where the EAP DH-CHAP authentication is unsuccessful, the conversation may appear as follows:

```
Authenticatee                  Authenticator
------------------------------ ------------------------------------
                               <- EAP-Request id=79 / Identity
EAP-Response id=79 / Identity
("Initiator")              ->
                               <- EAP-Request id=80 / DH-CHAP
                                  (g^x mod n, g, n, C_i, C_c)
EAP-Response id=80 / DH-CHAP
(g^y mod n, C_r)           ->
                               <- EAP-Failure id=81
```

8. Security Considerations

This entire draft is about security.

9. Open Issues

Group support. Need a list of Diffie-Hellman groups or group sizes that MUST be supported and a list of g and n values that SHOULD be used for various sizes of groups. This is also the case for SRP, for which Section 7.2 of [iSCSI] says:

> The strength of the SRP authentication method (specified in Chapter 13) is dependent on the characteristics of the group

being used (i.e., the prime modulus N and generator g). As described in [RFC2945], N is required to be a Sophie-German prime (of the form N = 2q + 1, where q is also prime) and the generator g is a primitive root of GF(n). In iSCSI authentication, the prime modulus N MUST be at least 768 bits.

Upon receiving N and g from the Target, the Initiator MUST verify that they satisfy the above requirements (and otherwise, abort the connection). This verification MAY start by trying to match N and g with a well-known group that satisfies the above requirements. Well-known SRP groups are provided in [SEC-IPS].

A better approach may be to explicitly require support and use of specific groups in order to avoid the need to test for N being a Sophie-German prime and g being a primitive root of GF(n).

The current design of bi-directional DH-CHAP protects responders from rogue initiators, but not vice-versa. This could be changed by having the responder (target) authenticate first rather than the initiator. It's not clear that this makes a significant difference, as a successful dictionary attack against a responder secret can be used to impersonate the responder to the initiator to attack the initiator directly or obtain information to mount a dictionary attack on the initiator's secret.

10. Change Log

10.1 -00 to -01

- Changed author from "Hain" to "Black" in pp.2+ footers.

- Rewrote section 6.4 to incorporate explanation and example of reflection attack from Paul Koning.
- Removed "(which will generally lead to an authentication failure)" from the Overview, as it is not true of impersonation attacks on one-way authentication.
- Strengthened the warning in Section 6.5 that IPsec needs to be used when active attacks against DH-CHAP are a concern.
- Lots of editorial changes.

References

[2284bis] L. Blunk, J. Vollbrecht, B Aboba, "Extensible Authentication Protocol (EAP)," draft-ietf-pppext-rfc2284bis-03.txt, Work in Progress, 2 April 2002.

[EAP-SRP] J. Carlson, B. Aboba, H. Haverinen, "EAP SRP-SHA1 Authentication Protocol", draft-ietf-pppext-eap-srp-03.txt, Work in Progress, July 2001.

[iSCSI] Satran, J., et. al., "iSCSI", draft-ietf-ips-iscsi-12.txt, Work in Progress, March 2002.

[RFC 1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August, 1996.

[RFC 2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March, 1997.

[RFC 2284] L. Blunk, J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)," RFC 2284, March 1998.

[RFC 2869] C. Rigney, W. Willats, P. Calhoun, "RADIUS Extensions", RFC 2869, June 2000.

[RFC 2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September, 2000.

[Schneier] Schneier, B., "Applied Cryptography, Second Edition", New York: John Wiley & Sons, Inc., 1996.

Acknowledgements

A combination of Diffie-Hellman with CHAP was originally suggested by Steve Bellovin. The augmentation approach of concatenating the DH key to the CHAP challenge was suggested by Uri Blumenthal. Steve Senum contributed the text on EAP in Section 7.1 and its subsections. The explanation of reflection attacks and the example in Section 6.4 are largely based on Paul Koning's discussion of this topic on the IPS mailing list. Improvements have resulted from comments on earlier versions of this draft by a number of people, including Ofer Biran and Mark Bakke. Additional comments on various topics from the IPS WG mailing list have been incorporated.

Author's Address

David L. Black
EMC Corporation
42 South Street Phone: +1 (508) 249-6449
Hopkinton, Mass., 01748, USA Email: black_david@emc.com

Matchmaker:
An Interface Specification Language for Distributed Processing

Michael B. Jones
Richard F. Rashid
Mary R. Thompson

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Matchmaker, a language used to specify and automate the generation of interprocess communication interfaces, is presented. The process of and reasons for the evolution of Matchmaker are described. Performance and usage statistics are presented. Comparisons are made between Matchmaker and other related systems. Possible future directions are examined.

Keywords

Remote Procedure Call, Interprocess Communication, Object-Oriented Languages, Multi-Targeted Compiler, Interface Specification Language, Distributed Systems.

1. Introduction

One of the thorniest problems in building a distributed system is how to interface distributed system components. The earliest distributed systems required programs to directly manage a communication protocol on an I/O channel. Work on message-based systems often led to a style of interprocess interaction which stressed communication flexibility over interface correctness or ease of programming. Interfaces were effectively implemented in "message assembly language" as message records were explicitly packed and unpacked by user written code.

In recent years a number of distributed programming languages have been proposed. Design and implementation considerations have often been driven by abstract issues rather than the concrete requirements of a distributed system. Designing a single distributed programming language often ignores the fact that many applications are already written in existing languages such as C, Pascal and in the case of AI applications, LISP. Frequently such languages stress simple client/server communication and ignore the requirements of real-time system services.

Rather than being another distributed programming language, Matchmaker is an interface specification language for use with existing programming languages. It provides:

- a language for specifying object-oriented remote procedure call (RPC) interfaces between processes executing on the same machine or within the SPICE network,
- a multi-targeted compiler which converts these specifications into interface code for each of the major languages used within the SPICE environment, including C, PERQ Pascal [1], COMMON LISP [14] and Ada [4]. This code provides communication, runtime support for type-checking, synchronization and exception handling.

Matchmaker was started in 1981 as part of the CMU distributed personal computing project (SPICE [3]). It was built at first to automate some of the coding for the Accent[1]

[1] Accent is a tradmark of Carnegie-Mellon University

operating system kernel [11] message interface, which forms the basis of the SPICE environment. It has evolved significantly during the last three years in its syntax, data representation semantics and communication semantics. At each point of change, decisions about the new Matchmaker design and implementation were driven by specific requirements of programmers in the SPICE environment.

Over the years, Matchmaker has proven to be a valuable tool. It has:

- eased implementation and improved the reliability of distributed programs by detaching the programmer from concerns about message data formats, operating system peculiarities and specific synchronization details,
- improved cooperation between system programmers working in different languages,
- enhanced system standardization by providing a uniform message level interface between processes,
- provided a language rich enough to express any data structure which can both be efficiently represented in messages, and reasonably represented in all target languages,
- reduced the cost of reprogramming interfaces in multiple languages whenever a program interface is changed.

Matchmaker provides a wide range of synchronization semantics ranging from synchronous remote procedure call to asynchronous message-style communication. A programmer can usually change the synchronization part of the Matchmaker specification without affecting the code which uses that interface.

Today, Matchmaker interfaces define all interprocess communication in the SPICE environment which consists of over 150 PERQ[2] computers communicating on an internetwork of several 3MHz and 10MHz Ethernets. Matchmaker is also used to specify and implement interprocess communication interfaces between PERQs and the CMU CS Department's 40 VAX[3] computers, which run a modified version of Berkeley 4.1bsd UNIX[4] [12] supporting Accent-style message communication. In all, Matchmaker has been used as the distributed programming support environment for over 500,000 lines of code written in four major languages. Matchmaker has evolved from a simple programming aid into the effective definition of interprocess communication within the SPICE environment.

In this paper we will discuss the Matchmaker language -- its syntax, data representation and communication semantics, and its implementation. We will also examine the issues which forced many of the important decisions in the Matchmaker design.

2. Language Overview

The computational model for Matchmaker consists of processes communicating with one another via messages. Messages are sent to communication ports. Accent ports, and rights to receive messages from specific ports, can be sent between processes in messages.

Ports also serve in a dual role as capabilities for objects. Matchmaker interfaces define operations upon those objects. Every remote procedure call specifies a destination port for the request. Thus, the ports may be viewed as tokens for instantiations of objects, and RPC requests to ports may be viewed as invocations of operations upon objects. Such an identification logically makes every Matchmaker request to a port an operation on the object represented by that port.

The syntax of Matchmaker specifications is fairly close to the Pascal or Ada specifications for the analogous objects. Constants of various types can be declared, new data types can be constructed from built-in types (within certain constraints), and remote procedures can be declared with a syntax fairly similar to Pascal procedures or functions. The invocation of a remote procedure on a port in a given target language usually consists of a procedure call, with that port as the first procedure parameter.

The built-in data types provided by Matchmaker are: **Boolean, Character, Signed** and **Unsigned Integers** of various

[2] PERQ is a trademark of Perq Systems Corporation

[3] VAX is a trademark of Digital Equipment Corporation

[4] UNIX is a trademark of AT&T Bell Laboratories

bit sizes, **Integer SubRanges**, **Strings**, **Communication Ports**, and **Reals**. New data types can also be constructed with some restrictions. Type constructor functions supported are: **Records**, fixed and variable-sized **Arrays**, **Enumerations**, **Pointers** to the above types, and certain kinds of **Unions**.

Representations for remote procedure arguments in messages are chosen by the Matchmaker compiler. Each message is assigned a unique id by Matchmaker, which is used at run-time to identify messages for a given interface. Once the message has been identified, the types of all fields within it are also known, since messages are strongly typed by Matchmaker at compile-time.

Certain semantic restrictions are placed upon the data types which can be declared to allow efficient passing of arguments in messages. In particular, pointers, variable-sized arrays, and unions can only occur in top-level remote procedure call declarations, and may not be used when constructing other types.

Several semantically different kinds of remote procedure call interactions can be specified in Matchmaker. The process normally initiating an operation is called the *client process*, and the process normally receiving requests is called the *server process*. The RPC paradigms provided are:

- **Remote Procedure**: Generates code for a client process to send a request to a server, and to receive reply parameters back from the server. Timeout values can be specified, and the reply wait can be made asynchronous as well.

- **Message**: Generates code for a client process to send a single request message to a server without a reply.

- **Server_Message**: Generates code for a server process to send a single message to a client process.

- **Alternate_Reply**: Generates code for a server process to send a reply message back to a client process in response to a **Remote_Procedure** which is different than the normal reply message. **Alternate_Reply** messages are meant to be used for signaling exception conditions which occurred during execution.

Each of these varieties of calls except for Alternate_Reply takes a port to which to send the request as a parameter. Thus, "binding" is done dynamically on the basis of ports, and not by using some compile-time or link-time discipline.

3. Language Evolution

The Matchmaker program was originally conceived as a programming tool to simplify the sending and receiving of messages. It was planned to be a temporary expedient which would generate Pascal code until language intrinsics for sending and receiving messages could be added to the Pascal compiler. The intention to add interprocess communication support to our main programming languages was in line with contemporary distributed programming language proposals such as PLITS [5].

The original input to Matchmaker consisted of the names of the procedures to be generated and the list of parameters to each procedure. Associated with each parameter was an indication of the direction in which it was to be sent, the Pascal type of the parameter and the message-specific type description for the parameter. Since the only target language anticipated at that time was Pascal, the Pascal type declarations were imported into the generated code. This first version only generated simple synchronous remote procedure calls. Each generated procedure call would send a message to the server and then wait for a reply before returning to the client.

After several months of use it became apparent that the Matchmaker approach had several important advantages over the notion of adding language intrinsics to Pascal:

- The procedure-based form of Matchmaker generated calls made these interfaces easy to document and use.

- Such intrinsics would have to be added to each language which was to be used in the SPICE environment. Moreover, adding intrinsics would leave the project with the burden of supporting a non-standard version of each language.

- The input to Matchmaker could be developed into a language independent formal specification of the interfaces to the system servers.

This early version of Matchmaker also had its share of problems, however:

- The inflexible format of Matchmaker specifications made them difficult to use.

- Data type specification was awkward and too limited.

- The semantics of remote procedure call were too limited.

In response to these problems, the second version of Matchmaker allowed declaration of types in a Pascal-like syntax, the specification of some global message style options, and the specification of an arbitrary number of RPC interfaces. As Matchmaker was used to generate interfaces for more servers, variations on the remote procedure call were added. For example, the window management process was sent character strings to display on the screen. These messages did not require a reply message, so messages without replies were added. Some applications wanted to use remote function calls rather than procedure calls. Procedures that signaled their errors as exceptions were also added. Client and server processes were found to require completely asynchronous communication for some tasks. During this period, the evolution of Matchmaker was driven by specific demands made by the writers of server processes.

At this time, Matchmaker was widely used only by Pascal programmers and still required the inclusion of Pascal import files. The next major modification to the Matchmaker language came as a result of the need to use the Matchmaker specifications to generate C and LISP code. Since C is close to Pascal in style, it was possible to use Matchmaker to generate C code and to import the language types from C include files, instead of Pascal import files. The LISP implementors were not so fortunate. The usefulness of COMMON LISP was delayed by the need to hand code the LISP function to message interface for all the system servers. It was now obvious that a genuinely language independent specification was needed for the server interfaces.

The third and current version of the Matchmaker language fulfills this requirement. A Matchmaker specification now includes complete descriptions of the types of every argument that is passed. Matchmaker generates the target language (Pascal, C, LISP, etc.) type declarations to be imported into the generated code. The Matchmaker specification for a client/server interface is written in a formal language that is approximately as readable as Pascal type and procedure declarations. This specification is both the documentation of the interprocess interfaces, and the source code which is compiled into correct procedure calls and type declarations for the target language. The Matchmaker compiler is internally structured to allow the addition of code generators for other languages as they are added to the SPICE environment.

This same version of Matchmaker also includes enhancements which allow a fine grain of control over the message send/receive options. While older Matchmaker implementations made various assumptions about the manner in which messages were to be sent and received, it is now possible to control all such parameters, both statically and dynamically.

4. Usage and Performance

Given the extensive usage of Matchmaker within the SPICE system, there are a number of interesting statistics which are available on the use and performance of Matchmaker interfaces. The figures below were gathered from the PERQ Pascal interfaces to the standard SPICE server interfaces, including the Accent kernel.

Static Usage:

Number of Interfaces	15
Total Calls Declared	268
Total Asynchronous Requests	67
Total Alternate_Replys	7

Dynamic Usage:

No exact figures are available, but it is known that far more asynchronous (unacknowledged) calls take place than synchronous ones. This is due to the fact that most I/O activities such as screen, mouse, low-level keyboard, and Ethernet I/O are handled asynchronously.

Avg. Code Bytes Per Call:

	Client	Server
Min. (Accent Kernel)	146	165
Max. (Filesystem)	246	242
Avg.	212	181

It is significant to note that server interface code is usually smaller than the corresponding client code. This directly corresponds to the fact that more parameters tend to be passed into calls than are returned by them.

% Matchmaker Code by Size in Servers:

Min. (Authorization Server)	1.4%
Max. (Filesystem)	23%
Kernel	6.5%

The relative size of the interface code varies with the number of routines provided, the number of arguments passed, and the amount of processing requested by each call. The Kernel and Authorization Server each use one relatively simple interface, and respond directly to requests from clients. The filesystem, on the other hand, is implemented as a set of co-operating processes distributed across several machines, using several different interfaces. Hence, filesystem processes are clients of one another via inter-filesystem interfaces, as well as being servers, thus explaining the high percentage of interface code.

% Matchmaker Code by Size in Clients:

The percentage of total code in client processes which is Matchmaker code is not especially useful, since for all standard SPICE servers, client interfaces are imported from shared runtime libraries.

Avg. Message Passing Overhead with Arguments:

Bare Kernel Send & Receive	2.5 ms
Matchmaker Overhead	0.6 ms
Total Msg Passing Time	3.1 ms

Thus, Matchmaker interfaces account for ~ 19% of message passing time, when packing and unpacking the arguments into messages is included.

Message Passing Frequency:

Avg.	25 to 30 msgs / sec
Max. Ever Observed	110 msgs / sec

Msg Passing Time as % Total Time:

~ 30 msgs / sec
x 3.1 ms / msg
= ~ 9% total time spent in msg passing

Matchmaker Overhead as % Total Time:

~ 30 msgs / sec
x 0.6 ms Matchmaker overhead / msg
= ~ 1.8% total time in Matchmaker overhead

The currently generated Matchmaker code is known to be inefficient. Yet, it is important to note, as shown by the above statistics, that actual Matchmaker overheads in message passing have no perceivable effect upon system performance. If they were eliminated entirely, it would not be noticed.

Although all communication in SPICE is via messages and Matchmaker, we know of no normal system activity which is dominated by message passing time. Even at the maximum observed message passing rate, roughly 2/3 of the total time would still be used for other things. This is easily explained; it almost always takes longer to process the information passed in messages than the time it took to transmit it, given Accent's efficient message implementation.

Throughout several years of use, Matchmaker has permitted distributed applications to be built with nearly the same ease as single-process applications. Two examples illustrate this point:

- The SPICE window manager was originally written and debugged as a stand-alone application and then converted into a server process using Matchmaker. The conversion process required virtually no change to the underlying structure of the program.

- A set of autonomous file servers were converted into a distributed filesystem in less than a week by designing and implementing an appropriate Matchmaker interface between them.

5. Comparisons

Matchmaker can be most directly compared with Nelson's Diplomat [10] and the Cedar Lupine system [2]. Matchmaker, Diplomat and Lupine can all be described as remote procedure call stub generators. Both of these Xerox systems, however, were built around a single programming language. Lupine, in particular, uses the existing Mesa [9] interface modules as the basis for generating the remote procedure call stubs.

Matchmaker evolved during roughly the same period as Diplomat and Lupine. Matchmaker differs from these efforts in that it: is an external specification language, supports multiple languages in a heterogeneous machine and operating system environment, provides for a wide class of synchronization semantics in addition to remote procedure call, and supports an object-oriented computational model. Matchmaker is also unique in that it is the sole interface language for both local and network communication.

Matchmaker can also be compared with earlier attempts in the RIG [7] system to build generic interprocess interfaces. RIG provided a "Call" function which took as its arguments the object to be operated on (represented as a RIG process-port pair), the function to be invoked (message identifier), and the arguments. Matchmaker interfaces in contrast to the RIG approach are type checked, handle multiple languages in the style appropriate to that language and allow for greater flexibility in defining the information to be passed as part of a remote call.

Unlike the Argus's Actions and Guardians protocol [8], Matchmaker does not provide for atomic transactions. The nearest that a server can get to providing atomic transactions is to provide **Remote_Procedure** interfaces, with reply status values, and reply timeout values that cause blocking until a reply is received. These actions can then be known to be atomic in some cases. If the server cannot receive the message, the reply code is set to "Failure" and no action takes place. Likewise, if the server is reached, but can not successfully carry out the request, it will return a "Failure" code and abort the entire transaction. However, the hard case where a server is reached and then crashes before it completes the transaction, either leaves the client permanently blocked waiting to receive a reply, or returns a "Timeout" status, depending upon the options selected.

Unlike systems that are written entirely in one strongly typed language such as Argus/CLU and the Xerox systems, Matchmaker's type checking may be compromised by the language that invokes its interfaces. Matchmaker runtime code checks the types of the arguments that are extracted from messages but it must rely on its implemention language (Pascal, LISP, C, etc.) to guarantee the integrity of the values passed to it as parameters.

6. Future Directions

As with any evolving system, there is still substantial room for improvement in Matchmaker.

Matchmaker does not enforce a robust implementation of interprocess communication. Rather, it allows the implementer of a server process to choose from the underlying primitive communication paradigms provided, and to easily provide an RPC interface to the user.

A synchronous remote procedure call does not currently terminate when the target server process terminates abnormally. Instead, an exception or timeout is generated which must be handled by the client. It was originally felt that this was an adequate solution to the problem, but as more and more naive programmers use the system for developing their own applications, it has become apparent that handling such conditions can be difficult.

As a result, system work is underway to allow for a synchronous error return to be provided when a server crashes during the execution of a remote request. This support should be an appropriate mechanism for implementing truly atomic remote procedure calls. Likewise, enhancements were recently added which allow a fine grain of control over the message send/receive options. With these improvements a careful server implementer should be able to write a robust and transparent server interface that requires no particular sophistication on the part of the user of the interface.

The future direction of Matchmaker will probably be influenced by the development of SPICE applications that are implemented as closely co-operating servers distributed over more than one machine. Both the Sesame File System [6] and the TABS [13] Distributed Transactions manager are currently being implemented in this manner. The issues of robustness in the face of a remote server failure, and guaranteed response to the original client are being addressed by these servers.

7. Conclusions

Matchmaker is not a new distributed programming language; it is not a radical departure from existing techniques in the design and implementation of programming languages. Instead, Matchmaker is an important tool for distributed programming which has been evolving and in use for over three years. It has proven valuable and simple to use. It allows a server to automatically be accessible from clients written in any of the supported languages, regardless of the language in which the server is written. It permits distributed applications to be built with nearly the same ease as single-process applications.

Probably Matchmaker's greatest value is that it has become, in effect, *the* working definition of inter-domain communication in the SPICE system. Since it automates the implementation of RPC on top of messages, it is conceivable that different Matchmaker code generators could implement a similar form of RPC on a different communication medium, with almost no change to the client or server code involved.

Through use in real distributed systems, Matchmaker has succeeded in proving itself a useful tool for creating interprocess interfaces in a very demanding distributed environment.

I. Example Specification

The text which follows is a fictional Matchmaker interface specification for a "display server" process.

```
Interface Screen = 15000;       ! Base Msg ID is 15000

Constant
    Max_X       = 132;
    Max_Y       = 40;

    Inverted    = true;
    Normal      = not Inverted; ! A constant expression

Type
    Screen_Array = packed array [Max_X * Max_Y] of Character;

    Char_Vector  = † packed array [*] of Character;

    Screen_State = record
                     x          : byte;
                     y          : byte;
                     Reverse    : boolean;
                   end record;

    Screen       = port;   ! Port used for screen token

Message DisplayChars(
                        : Screen;
        x               : byte;
        y               : byte;
        chars [num]     : Char_Vector;  ! Note size parameter
        ) : No_Value;

Message PutChar( : Screen; c : Character) : No_Value;

Message ClearScreen( : Screen) : No_Value;

Remote_Procedure GetWholeScreen(
                        : Screen;
   out  ScreenArray     : Screen_Array;
   out  Current_X_Size  : byte;
   out  Current_Y_Size  : byte;
        ) : GR_Value;

Remote_Procedure SwapScreenState(
                        : Screen;
  inout State           : Screen_State;
        ) : No_Value;

Alternate_Reply No_Such_Screen;

End Interface
```

II. Example Matchmaker Output

This appendix contains the Matchmaker source and generated PERQ Pascal output for the client side of one call in an actual interface used in SPICE. The call presented is relatively simple in comparison to many of the calls used by the system. Essentially, it sends a communication port to a server, and receives a new port back from that server in a reply message.

The Matchmaker specification for the call is:

```
Remote_Procedure CopyEnvConnection(
                                : Port;
              OldConnection     : Port;
          out NewConnection     : Port_Ownership
    ) : GR_Value;
```

The generated Pascal code implementing the client interface depends upon the following data structures, which define Accent messages and type descriptors within messages. They are presented here without explanation (see [11] for details) in order to clarify the code to follow.

```
TypeType = packed record
             case integer of
               1: ( TypeName       : Bit8;
                    TypeSizeInBits : Bit8;
                    NumObjects     : Bit12;
                    InLine         : boolean;
                    LongForm       : boolean;
                    Deallocate     : boolean
                  );
               2: ( LongInteger : long)
           end;

Port     = long;

Msg      = record
             SimpleMsg  : boolean;
             MsgSize    : long;
             MsgType    : long;
             LocalPort  : Port;
             RemotePort : Port;
             ID         : long;
           end;
```

Finally, the generated client code for sending the request, and receiving the reply is as follows. ("`ReplyPort`" is an Own variable; "`{$IFC ... }`" is conditional compilation; "`WordSize`" is like "`sizeof`" in C.)

```
Function CopyEnvConnection(
           ServPort     : Port;
           OldConnection        : Port;
         var NewConnection      : port
    ): GeneralReturn;

type
    MyMessage = record
        head            : Msg;
        IPCNam2         : TypeType;
        Arg2            : port;
    end;

type
    RepMessage = record
        head            : Msg;
        RetCodeType     : TypeType;
        RetCode         : Integer;
        IPCNam3         : TypeType;
        Arg3            : port;
    end;

  var
    MyMsg       : MyMessage;
    RepMsg      : RepMessage;
begin

    with MyMsg.head do
    begin
        SimpleMsg := false;
        MsgSize := WordSize(MyMsg)*2;
        MsgType := NORMALMSG;
        RemotePort := ServPort;
        LocalPort := ReplyPort;
        ID :=     1604;
    end;
    with MyMsg do
    begin
{$IFC FastAssign then}
        IPCNam2.LongInteger := #2000220006;
        {$ELSEC}
        IPCNam2.InLine :=   TRUE;
        IPCNam2.Deallocate :=  FALSE;
        IPCNam2.LongForm :=  FALSE;
        IPCNam2.TypeName := TypePt;
        IPCNam2.TypeSizeInBits := 32;
        IPCNam2.NumObjects := 1;
        {$ENDC}
        Arg2 := (OldConnection);
    end;
    with RepMsg.head do
    begin
        MsgSize := WordSize(RepMsg)*2;
        LocalPort := ReplyPort;
    end;
    GR := Send(MyMsg.head,0,WAIT);
    if GR <> Success then
    begin
        CopyEnvConnection := GR;
        exit(CopyEnvConnection);
    end;

    GR := Receive(RepMsg.head,0,LOCALPT,RECEIVEIT);
    if GR <> Success then
    begin
        CopyEnvConnection := GR;
        exit(CopyEnvConnection);
    end;
    with RepMsg do
    begin
        if head.ID <>     1704 then
        begin
            CopyEnvConnection := BADREPLY;
            exit(CopyEnvConnection);
        end;
        {$IFC TypeCheck THEN}
        if RetCodeType.TypeName <> TYPEINT16 then
        begin
            CopyEnvConnection := BADREPLY;
            exit(CopyEnvConnection);
        end;
        {$ENDC}
        CopyEnvConnection := RetCode;
        {$IFC TypeCheck THEN}
        if IPCNam3.TypeName <> TypePtOwnership then
        begin
            CopyEnvConnection := BadReply;
            exit(CopyEnvConnection);
        end;
        {$ENDC}
        NewConnection := (Arg3);
    end;
end;
```

III. Matchmaker Language Syntax

The following is a syntax description of the MatchMaker language. Conventions used are as follows:

Double quotes (" ") denote literal tokens.
Square brackets ([ ]) denote optional productions.
Braces ({ }) are used to enclose a group of productions.
Three periods (. . .) denote optional repetition.
Vertical bars (|) separate choices between productions.
Parens (( )) are used to enclose comments.

Interface and Options Definitions

```
Specification
     ::= Interface_Spec
     ::= Types_Spec

Interface_Spec
     ::= Interface_Decl [Options_Decl]... [Data_Decl]...
         [Msg_Decl]... "End" "Interface"

Types_Spec
     ::= Types_Decl [Options_Decl]... [Data_Decl]...
         "End" "Types"

Interface_Decl
     ::= "Interface" Interface_Name "=" Msg_ID_Base ";"

Types_Decl
     ::= "Types" Interface_Name ";"

Interface_Name
     ::= Identifier

Msg_ID_Base
     ::= Integer_Constant

Options_Decl
     ::= "Options" {Option_Decl ";"}...

Option_Decl
     ::= Msg_Options
     ::= Protocol_Options
     ::= Ports_Options

Protocol_Options
     ::= "Protocol_Version" "=" Integer_Constant

Ports_Options
     ::= "Local_Ports" "=" {Integer_Constant | "*"}
     ::= "Ports_Backlog" "=" Integer_Constant
```

DataType Definitions

```
Data_Decl
     ::= Use_Decl
     ::= Type_Decl
     ::= Constant_Decl

Use_Decl
     ::= "Use" Single_Use...

Single_Use
     ::= Interface_Name "From" File_Name ";"

File_Name
     ::= String_Constant

Constant_Decl
     ::= "Constant" Single_Constant...

Single_Constant
     ::= Constant_Name "=" Constant_Expr ";"

Constant_Name
     ::= Identifier

Type_Decl
     ::= "Type" Single_Type...

Single_Type
     ::= Type_Name "=" Type_Specification ["." Type_Option]...
     ";"

Type_Name
     ::= Identifier

Type_Option
     ::= "TypeType" "=" Integer_Constant
     ::= "Deallocate" ["=" Boolean_Constant]
     ::= "NoDeallocate"
     ::= "Element_Size" "=" Integer_Constant
     ::= "Element_Count" "=" Integer_Constant

Type_Specification
     ::= Type_Name
     ::= Builtin_Type
     ::= Array_Type
     ::= Record_Type
     ::= Pointer_Type
     ::= Enumeration_Type
     ::= Union_Type

Builtin_Type
     ::= "Boolean"
     ::= "Character"
     ::= "Real"
     ::= Integer_Type
     ::= String_Type
     ::= Port_Type

Integer_Type
     ::= "Unsigned" ["[" Integer_Constant "]"]
     ::= "Signed" ["[" Integer_Constant "]"]
     ::= Subrange_Type
     ::= "Long"
     ::= "Short"
     ::= "Byte"

Subrange_Type
     ::= Integer_Constant ".." Integer_Constant

Port_Type
     ::= "Port"
     ::= "Port_Send"
     ::= "Port_Receive"
     ::= "Port_Ownership"
     ::= "Port_All"

String_Type
     ::= "Perq_String" ["[" Integer_Constant "]"]

Array_Type
     ::= [Packing] "Array" "[" Array_Size "]" "Of"
     Type_Specification

Array_Size
     ::= Integer_Constant
     ::= "*"

Packing
     ::= "Packed"
     ::= "Unpacked"

Record_Type
     ::= [Packing] "Record" Record_Component... "End" "Record"

Record_Component
     ::= Field_Identifier ":" Type_Specification ";"

Field_Identifier
     ::= Identifier

Pointer_Type
     ::= "↑" Type_Specification

Enumeration_Type
```

```
    ::= "(" Enum_List ")"

Enum_List
    ::= Enum_Element ["," Enum_Element]...

Enum_Element
    ::= Enum_Name ["=" Integer_Constant]

Enum_Name
    ::= Identifier

Union_Type
    ::= "Union" "<" Union_Selector_Type ">" "Of"
        Union_Component... "End" "Union"

Union_Selector_Type
    ::= Type_Specification

Union_Component
    ::= Union_Tag ":" "(" [Record_Component] ")" ";"

Union_Tag
    ::= Constant_Expr
    ::= "Otherwise"
```

Message Definitions

```
Msg_Decl
    ::= Msg_Code_Decl
    ::= Msg_ID_Decl

Msg_Code_Decl
    ::= Msg_Body ["," Msg_Options]... ";"

Msg_Options
    ::= Msg_Param_Key "=" Integer_Constant

Msg_Body
    ::= "Message" Arg_List ":" Msg_Result
    ::= "Remote_Procedure" Arg_List ":" Msg_Result
    ::= "Server_Message" Arg_List
    ::= "Alternate_Reply" [Arg_List]

Msg_Result
    ::= Special_Result
    ::= Arg_Type

Special_Result
    ::= "GR_Value"
    ::= "No_Value"

Arg_List
    ::= "(" Msg_Arg [";" Msg_Arg]... ")"

Msg_Arg
    ::= Data_Arg
    ::= Special_Arg

Data_Arg
    ::= [Arg_Direction] Data_Arg_Spec;

Arg_Direction
    ::= "In"
    ::= "Out"
    ::= "InOut"

Data_Arg_Spec
    ::= Simple_Arg_Spec
    ::= Variable_Arg_Spec
    ::= Union_Arg_Spec

Simple_Arg_Spec
    ::= Arg_Name ":" Arg_Type

Variable_Arg_Spec
    ::= Arg_Name "[" Arg_Cnt_Name "]" ":" Arg_Type
    ::= "[" Arg_Cnt_Name "]" Arg_Name ":" Arg_Type

Union_Arg_Spec
    ::= Arg_Name "<" Selector_Name ">" ":" Arg_Type
    ::= "<" Selector_Name ">" Arg_Name ":" Arg_Type

Special_Arg
    ::= Special_Usage Arg_Name ":" Arg_Type
    ::= ":" Arg_Type

Special_Usage
    ::= Port_Usage_Key
    ::= Msg_Param_Key

Port_Usage_Key
    ::= "RemotePort"
    ::= "LocalPort"

Msg_Param_Key
    ::= "MsgType"
    ::= "ReplyType"
    ::= "Send_Option"
    ::= "Send_Timeout"
    ::= "Receive_Timeout"

Arg_Cnt_Name
    ::= Arg_Name

Selector_Name
    ::= Arg_Name

Arg_Name
    ::= Identifier

Arg_Type
    ::= Type_Name ["." Type_Option]...

Msg_ID_Decl
    ::= "Skip_ID" ";"
    ::= "Next_ID" "=" Integer_Constant ";"
```

Expression Syntax

```
Constant_Expr
    ::= Or_CTCE (Valid types context dependent)

Integer_Constant
    ::= Adding_CTCE (Must be integer valued)

Boolean_Constant
    ::= Or_CTCE (Must be boolean valued)

Character_Constant
    ::= Primary_CTCE (Must be character valued)

String_Constant
    ::= Primary_CTCE (Must be string valued)

Enumeration_Constant
    ::= Primary_CTCE (Must result in a declared Enum_Name
    identifier)

Or_CTCE
    ::= And_CTCE ["Or" And_CTCE]...

And_CTCE
    ::= Not_CTCE ["And" Not_CTCE]...

Not_CTCE
    ::= ["Not"] Relational_CTCE

Relational_CTCE
    ::= Equality_CTCE
        [ {">" | ">=" | "<=" | "<"} Equality_CTCE ]...

Equality_CTCE
    ::= Adding_CTCE [ {"=" | "<>"} Adding_CTCE]...

Adding_CTCE
    ::= [{"+" | "-"}] Multiplying_CTCE
        [{"+" | "-"} Multiplying_CTCE]...

Multiplying_CTCE
    ::= Primary_CTCE [ {"*" | "/" | "Mod"} Primary_CTCE]...

Primary_CTCE
    ::= Identifier
    ::= Constant_Lexeme
    ::= "(" Or_CTCE ")"
```

Lexical Definitions

```
Constant_Lexeme
     ::= Octal_Literal
     ::= Decimal_Literal
     ::= String_Literal
     ::= Character_Literal
     ::= Boolean_Literal

Octal_Literal
     ::= "#" followed by a non-empty octal digit string.

Decimal_Literal
     ::= A non-empty decimal digit string.

String_Literal
     ::= A character string enclosed in double quotes.  A
     double quote in a string must be doubled.

Character_Literal
     ::= A character enclosed in single quotes.  A single quote
     in a character literal must be doubled.

Boolean_Literal
     ::= "True"
     ::= "False"

Identifier
     ::= A string composed of letters, digits and the
     underscore character, not starting with a digit.
     Identifiers are matched in a non-case-sensitive manner.

Comment
     ::= At any lexical break, comments can be inserted as:
         "!" Arbitrary comment text <End_Of_Line>
```


Acknowledgments

The authors would like to express their thanks to the following people who helped in the design, evolution, and implementation of Matchmaker: Jeff Eppinger, Joe Ginder, Jim Large, Rob MacLachlan, Doug Philips, Keith Wright, and Mike Young. Thanks also go to Bob Fitzgerald, who provided some of the statistics for this article.

This research was sponsored by the Defense Advanced Research Projects Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

References

1. Miles Bartel, Michael Kristofic. PERQ Pascal Extensions. In *PERQ Software Reference Manual*, Three Rivers Computer Corporation, 1982.

2. Birrell, A. D. and Nelson, B. J. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems 2*, 1 (February 1984), 39-59.

3. . Proposal for a joint effort in personal scientific computing. Tech. Rept. , Computer Science Department, Carnegie-Mellon University, August, 1979.

4. *Reference Manual for the Ada Programming Language.* July 1982 edition, Dept. of Defense, Ada Joint Program Office, Washington, DC, 1982.

5. Jerome A. Feldman. "High Level Programming for Distributed Computing." *Comm. of the ACM 22*, 6 (June 1979), 353-368.

6. Michael B. Jones, Richard F. Rashid, Mary Thompson. Sesame: The Spice File System. Carnegie-Mellon University, October, 1982. Internal Document

7. Keith A. Lantz, Klaus D. Gradischnig, Jerome A. Feldman, Richard F. Rashid. "Rochester's Intelligent Gateway." *Computer* (October 1982), 54-68.

8. Liskov, B. and Scheifler, R. Guardians and actions: Linguistic support for robust, distributed programs. Proceedings Ninth ACM SIGACT-SIGOPS Symposium on Principles of Programming Languages, ACM, January, 1982, pp. 7-19.

9. J.G. Mitchell, W. Maybury, R. Sweet. Mesa Language Manual. Xerox Research Report CSL-79-3, Xerox Research Center, Palo Alto, CA, 1979.

10. Bruce Jay Nelson. *Remote Procedure Call.* Ph.D. Th., Carnegie-Mellon University, May 1981.

11. Rashid, R. F. and Robertson, G. Accent: A Communication Oriented Network Operating System Kernel. Proceedings of the 8th Symposium on Operating Systems Principles, December, 1981, pp. 64-75.

12. D. Ritchie. "The Unix Time-Sharing System." *CACM 17*, 7 (July 1974), 365-375.

13. Alfred Z. Spector, Jacob Butcher, Dean S. Daniels, Daniel J. Duchamp, Jeffrey L. Eppinger, Charles E. Fineman, Abdelsalam Heddaya, Peter M. Schwarz. Support for Distributed Transactions in the TABS Prototype. Proceedings of the 4th Symposium on Reliability In Distributed Software and Database Systems, October, 1984. Also available as Carnegie-Mellon Report CMU-CS-84-132, July 1984.

14. Guy L. Steele Jr.. *COMMON LISP: The Language.* Digital Press, 1984.

The Tiger Project

Bill Bolosky

One day in early 1993 the MSR Operating Systems Group was having lunch in Microsoft Building 9. As usual, the conversation varied over a number of topics. The Mariners chances in the coming season: bad. The weather: bad. The fact that Microsoft had recently been sued by Stacker over compression in DOS 6.0: bad.

Despite the gloom, something good did come of that lunch. Rick Rashid asked us a question: What would it take to build a system that could source, deliver and display high-quality video? Today, most websites include video and every computer and network can deliver it. At the time few people had heard of (let alone used) the web, and PCs were barely able to show low resolution, low frame rate, grainy AVI videos from their own disks. Ten Mbit/s Ethernet passed for a fast network.

Rick's question led to a set of projects that gave rise to innovations in a number of areas: video servers; small, real-time computers to be television "set top boxes;" and formats for video and audio data, the descendants of which live on to this day. At the time, however, this just seemed like an interesting thought exercise for an afternoon.

Video is something that doesn't scale with computer performance, it depends on people. That is, what looks good is based on how people see and hear, not what goes on inside a computer. Consequently, there's a minimum amount of storage, network and computation necessary to supply video. These amounts are easily within the capabilities of any computer (or smart phone) today, but in 1993 hardware was just on the edge of being sufficiently capable. A single server had just enough capacity to serve a few simultaneous streams of video and enough disk to store only a few movies. Bob Fitzgerald saw this and realized that the obvious design of trying to assign movies to servers wouldn't work well. The popularity of movies varies widely. If you make copies of popular movies you quickly run out of storage space, and if you don't, you don't have enough bandwidth available to serve the popular movies.

After a few days of thinking, Fitz came up with a solution to the problem. Say you have a group of servers which together have enough storage capacity to store all of your movies and have enough bandwidth to supply all of your customers. If somehow you could apply all of that bandwidth to any of the movies, you'd be done. Fitz's insight was that by chopping up each movie into one second chunks and storing the chunks across all of the servers, as long as the users were offset in time the entire system bandwidth could just as easily supply a single movie as different ones.

For example, say you had four servers. Monty Python and the Holy Grail (which must be in any self-respecting movie library) would be chopped up into several thousand one-second chunks. The first chunk would go on the first server, the second on the second, and so on. After you'd put one chunk on each server, you just wrap around, so in our example the fifth chunk goes on the first server, etc. Take the rest of the movies in your library and treat them similarly. When playing a movie, every second the data comes from a different server and all of the users move through the servers in lockstep, so a system that's not overloaded stays that way as long as no new load is added.

Spreading data like this is called "striping." That made it obvious that we should name the system after a striped animal, of which two came to mind: zebras and tigers. Rick observed that tigers eat zebras, and so the choice was made. The machines in a Tiger system that held and transmitted the video were called "cubs."

We built Tiger systems from ordinary desktop PCs and disks, combined with good network switching, rather than using high-cost, high-capacity, high-reliability servers. While we didn't realize it at the time, we were one of the first instances of what later became the canonical design point for scalable computing: building from many, fault-prone commodity parts. Today, all high-scale web services are built that way, but we predated web search engines (not just Google, but its predecessor, Inktomi), scale-out databases, content distribution networks, etc.

It turns out that there is some interesting math involved in modeling how systems like this respond as they're loaded up, some good distributed systems problems in providing fault tolerance and getting rid of managing the schedule centrally, some tricky operating systems work to make the IO path efficient, and even bigger challenges in getting the networking to happen smoothly with the (quite limited) equipment of the day. Between Fitz, John Douceur, Steve Levi, Yoram Bernet and me, Rick had assembled the right set of people to solve the challenge.

We quickly discovered one thing. Compared with the operating systems work we had been doing, video gives excellent demos. We bought hardware MPEG-1 video decoders because the PCs were not fast enough to decode 1.5 Mbit/s MPEG on their own CPUs. With the decoder boards we were able to do video that probably was equivalent to what you get now from a middle-of-the-road clip on YouTube. We set up a lab with a Tiger system and a few dozen clients and started inviting people to see the system running. We gave demos to many people including Andy Grove (then CEO of Intel), Paul Allen and Bill Gates. On seeing the

demo, Bill commented that he wasn't aware that computers could render video that looked that good. The Tiger project was the first time that Bill saw high-quality, full-screen video on a PC.

Recall that our design point was running on commodity hardware. That meant that we had to expect machines to fail. Consequently, Tiger was designed to be able to keep running even after the failure of one (or sometimes more than one) of its cubs. This made for great theater in demos, which Rick in particular enjoyed. We would set up a Tiger system in a rack and have the cubs store their video on external disk drives with activity lights. We would start one video stream running, and Tiger's striping pattern would be obvious because the lights would flash once per second along the row of drives. We would then add load to the system, more videos would appear on the bank of client screens and the drive lights would start flashing much more constantly. At that point, Rick would demonstrate the fault tolerance by either turning off one of the drives or just unplugging a cub. A few of the clients might glitch for a second, but most of them went on uninterrupted and very quickly service would be restored to all of them. We even had a bit of a Pete Townshend fantasy that would involve smashing a cub with a hammer or even a shotgun for some particularly major demo, but we could never bring ourselves to do it. (Or maybe Rick was just a little more grown up than the rest of us and vetoed the idea.)

In order to serve a decent amount of video with the very limited computers of the day we had to use bleeding edge hardware. For networking we used 155 Mbit/s ATM networks that we bought from FORE systems. This hardware was so new that we were getting boards with single-digit serial numbers scratched into them by hand. Early in the project we had a reoccurring problem where every once in a while data would be corrupted somewhere between the disk and the client. After much searching it turned out to be a problem with the ATM NICs. The boards had a memory address line that was inverted, so the board firmware had to XOR that bit of memory addresses with 1 before sending it to the DMA controller. However, the firmware did some math on the addresses after the XOR had been applied. The result of this was that sending from a buffer that included a group of physical pages that were contiguous and spanned a 16MB boundary resulted in sending the data from 16MB lower in the physical address space. It took weeks to find this and get FORE to fix their firmware. We also found bugs in Windows and the MPEG rendering hardware.

After we got Tiger working (and also developed a set-top box operating system and some apps like an electronic program guide) we decided to do some trials. The first one was run from Building 15 on the Microsoft campus to a nearby apartment building. This trial pointed out what turned out to be the fatal flaw in the whole Microsoft video strategy: we had a wonderful server and a great set-top operating system, but there was no practical way to get sufficient network bandwidth between them to deliver the video.

For the Building 15 trial we put together an amazing Rube Goldberg contraption. We partnered with the cable TV company and they wired the apartment building to have analog cable that carried higher frequencies than were being used by ordinary cable. That is, we got a plain old analog broadcast TV cable with a couple dozen extra channels. This, of course, wasn't really how interactive television was supposed to work. The intent was to have a high-bandwidth

digital connection all the way to the set-top box. So, we put our "set-top boxes" in the head end (server room) with the Tiger system and connected them directly to the ATM network. Each of them was attached to one of the extra TV channels and just broadcast its output on the cable. In the apartments, we built another device that connected to the head end over dialup. When a user wanted interactive television functions we allocated one of head end "set-top boxes" to him and tuned his TV to the channel assigned to his set-top box. Every click on the remote went through the phone line.

We did a second trial in Yokosuka, Japan. NTT (the local telephone company) wired an apartment building with fiber optic network cables and connected them to the ATM switch in the head end. This trial worked very well, but again pointed out that the expense to set up the network overwhelmed the value in video on demand.

After Yokosuka we turned Tiger into the Microsoft Netshow Theater Server product and shipped it on July 7, 1998. Very few if any customers bought it, and Microsoft tore down the product team and gave up on the idea entirely immediately after it released to manufacturing.

There are a number of lessons to draw from the Tiger experience. One is that at many levels the network is the problem. We went into the design worrying about load balancing, disk performance, scheduling, distributed systems issues, fault tolerance, real time scheduling, etc. All of those problems turned out to be simple relative to just reliably getting bits through the networking stack and hardware. At the larger level the product effort as a whole failed to get customers because there simply was no economical method to get video-bandwidth data from a data center to customers in the late 90s.

There's also a lesson in the Tiger experience for Microsoft's business strategy. When you think of internet video today YouTube and Netflix come to mind. YouTube was founded in 2005; Microsoft shipped Tiger 7 years earlier. Netflix started offering streaming service three years later still, in 2008. We could have had both of these businesses rather than just ceding the entire field to competitors if we had stuck with the product we had. It's not as if we didn't understand that ubiquitous broadband networking was coming. Nathan Myhrvold's "Roadkill on the Information Highway" memo was written in 1993. Anyone who used the interactive TV demo system realized that video on demand was a much better experience than renting video tapes. Yet we were sufficiently shortsighted to decide to give up entirely when the network wasn't ready in '98. Netflix had over $3B of revenue in 2011. While Microsoft overall has had a wonderful track record of sticking with products while the market and technology caught up to the vision (think of SQL Server, Bing and Xbox) in this instance we dropped the ball and the billions of dollars of revenue that went with it.

But at least we can say that we were there first.

Twenty-Five Years with Rick Rashid

Rich Draves

lthough I studied Math as an undergrad, I knew that I wanted to switch to systems in graduate school. CMU seemed to be a great fit and I vaguely recall talking to Rick while making that decision. I didn't realize until much later how unusual an environment it was. For example, CMU Computer Science supported all of its graduate students without regard for their research interests or advisor's funding situation.

My first advisor at CMU was actually Alfred Spector. We were working on the Camelot project, which was a distributed transaction processing system. I was working on the lower layers of the system and started spending most of my time interacting with the Mach guys. I remember Mike Young, one of the Mach grad students, suggesting that I switch to the Mach project. I felt like Mike was inviting me to join the "inner circle" and so I changed advisors and started working with Rick.

Rick was very hands-off as an advisor. He seemed to dislike formal meetings (this is still true) and we did not have regularly scheduled 1-1s or even project meetings. Rick would just drop by your office. You might look up to find him peering over your shoulder. If you saw a lot of Rick, it either meant that you were working on the hot topic of the moment that had his attention, or perhaps that he was worried about you. If you didn't see much of Rick, then it either meant that you were doing fine or that he had written you off.

Now in my own work, I don't take this to the extreme that Rick did. I will have regularly scheduled 1-1s with people. But I do try to avoid the overhead of meetings and I especially try to avoid micro-managing people.

One of the first things that I did on the Mach project was help out with the port to the IBM RT. I don't recall all the details, but Mach would very occasionally crash on a new version of the workstation. Most often, it was actually a random user process that would mysteriously

crash, with register values that made no sense. I remember that it took me a couple months to track down the problem: it still stands as the longest time I ever spent debugging a single problem. (I could work on other issues while waiting for this crash to reproduce, so while progress was slow I was not blocked.) Ultimately it turned out to be a bug in the context-switch code. There was a floating-point coprocessor and some state was not being saved properly, so a context-switch in the middle of a floating point operation could result in the thread restarting with bad register values, generally all zeroes. The net result was that the thread would start re-executing from main() and eventually crash.

That was fun to figure out, and I ended up focusing on the inter-process communication (IPC) subsystem in Mach and its relationship to context-switching and thread management. The IPC subsystem was old code inherited from the Accent operating system. I rewrote the subsystem, fixing various semantic issues in the interface and also improving performance. I still remember the "aha" moment when I made a connection between a programming language concept, continuations, and what I was trying to achieve in optimizing the IPC subsystem. I had learned about continuations as a freshman taking Harvard's introductory computer science class, which was styled after MIT's infamous 6.001 Scheme class. This connection between continuations, IPC, and thread management inside the Mach kernel became the basis for my thesis.

While working on Mach at CMU, I experienced first-hand several of Rick's traits. The first was his fearlessness in diving in to try out an idea. If the grad students were skeptical or unwilling to try something, then Rick would just prototype it himself. Rick would get it working well enough to demo, but invariably there would be various subtle or not-so-subtle problems, so in the end someone would end up having to rewrite his code. But Rick's point would have been made.

The second important trait we called the Rashid Reality Distortion Field. Mach always worked best in Rick's office. It never crashed, it achieved the best performance, and if his code had subtle concurrency issues (it always did), they never manifested while he was present. Another aspect of the Rashid RDF was Rick's persuasiveness. When talking with Rick, you always saw things his way. Contentious issues would seem obvious and straight-forward. After leaving Rick, doubts and uncertainty might creep back.

A third trait (or perhaps this was just another aspect of the RDF) was Rick's productivity. We had a special term for it: Rashid Time. Rick could do everything faster. We learned to convert from Rashid Time to normal time by doubling and going to the next larger unit. For example, three hours for Rick would be six days for someone else, or what Rick could do in one day would take someone else two weeks.

You might assume that when Rick left CMU to start Microsoft Research, he immediately turned around and started recruiting me to join him. That's not the case.

I was quite content as a graduate student and I could easily have stayed at CMU much longer. Living on a graduate student stipend didn't bother me—Pittsburgh was cheap and in any case I had a better-than-usual stipend thanks to the Hertz Foundation. So I was in no hurry to leave.

It was actually another CMU faculty member, HT Kung, who deserves the credit for getting me to leave. At the same time that Rick left for Microsoft, HT left CMU for Harvard, on a mission to revitalize the computer science department there. HT knew that I had done my undergrad work at Harvard and he twisted my arm hard to apply for a junior faculty position. If I was going to apply to Harvard, I figured that I should apply elsewhere as well and so I sent off my resume to a few companies, Microsoft and NeXT and some others. Avie had gone to NeXT and I had done a summer internship there a couple years earlier. I didn't really see myself as an academic so although the Harvard opportunity was very intriguing I quickly narrowed my options to Microsoft and NeXT.

Although I didn't know much about Microsoft and I was a little dubious that it would really support a research lab long-term, I decided to accept the Microsoft offer. This was largely a matter of my belief in Rick. I figured that if he decided to go to Microsoft then I should give it a try too.

A funny thing happened after I communicated this decision to NeXT. I remember hanging out with my housemates when one of them answered the phone. They handed me the phone with a very bemused expression. It turned out that it was Steve Jobs on the line. I had seen him in the distance at NeXT but this was my first time actually speaking with him. He proceeded to harangue me for 30 minutes about how I was making a mistake and I should reconsider and join NeXT. I was polite but my mind was made up.

Now the year prior I had started dating a girl, Martha Moran, who was one of the first employees of Transarc, Alfred Spector's startup company. Transarc was essentially productizing the Andrew File System and Camelot projects from CMU. (I still have very fond memories of AFS. Twenty years later, it seems that Microsoft still doesn't have anything close to AFS for Windows, in terms of functionality and performance.)

I negotiated with Microsoft to have them pay for some trips between Seattle and Pittsburgh to help maintain the long-distance relationship. This seems pretty silly in retrospect—much better to have negotiated for more stock, for example. But I didn't place much value on the stock option component of my offer. I though at best, if I were really lucky, Microsoft stock might double in five years. In fact it increased by about a factor of ten in five years. Perhaps Rick anticipated this but I sure didn't.

A year after starting at Microsoft I was able to convince Martha to leave Transarc and join me at Microsoft in Seattle. (Sorry Alfred!) We actually worked on the same project (Interactive TV) for a time, and even both reported directly to Rick because he was wearing two hats, running Microsoft Research and also managing the ITV project for Craig Mundie.

There was one annual personnel review where Rick reviewed both of us, and he gave Martha better marks than me. Rick was always very brief in his personnel reviews. I still remember one annual review in which Rick's feedback consisted in its entirety of four words: "You are doing fine." Now that I'm a manager and I have to write review feedback, I also tend towards brevity but I'm still working to achieve Rick's mastery of the art form.

Although Rick always said I was doing well, in the first couple years at Microsoft I never felt that way myself. I had left CMU without actually finishing my PhD thesis. All the research was done, but I hadn't written it up. And when I got to Microsoft, it was easy to put off working on the thesis and focus on exciting new projects instead. I was surprised that Rick never pressured me to finish my degree. He never even brought it up. I felt increasingly guilty for avoiding it and I would downgrade myself on the annual reviews, but Rick always ignored my input. Perhaps he was employing some kind of reverse psychology. In any case, after a couple years of this, I finally decided that I really did want my degree and I started writing the thesis. The final straw was when I thought about having kids, and having to explain to them why I had never finished my PhD after spending so many years on it.

It ended up taking me four months to write my thesis. Rick always said that he could write a PhD thesis in two weeks, so this proves the Rashid Time conversion factor of doubling and going to the next larger unit. As far as I could tell, Rick never read a single chapter (or word) of my thesis, but I graduated so it was good enough. Although starting early at Microsoft worked out well for me in the end, I always strongly advise graduate students to finish their thesis before starting their next job.

When I look back, it's clear that Rick had an enormous influence on my career and my life, from graduate school at CMU to Microsoft Research. I learned how to do research, how to build systems, and how to manage people from Rick. I still ask myself "What would Rick do?" when considering a difficult situation.

The Drawbridge Library OS: Windows as an Application

Galen C. Hunt

Microsoft Research

"There is nothing new under the sun, but there are a lot of old things we don't know."
-Ambrose Bierce, The Devil's Dictionary

This paper revisits an old approach to operating system construction, the *library OS*, in a new context. The idea of the library OS is that the personality of the OS on which an application depends runs in the address space of the application. A small, fixed set of abstractions connects the library OS to the host OS kernel, offering the promise of better system security and more rapid independent evolution of OS components.

We describe a working prototype of a Windows 7 library OS that runs the latest releases of major applications such as Microsoft Excel, PowerPoint, and Internet Explorer. We demonstrate that desktop sharing across independent, securely isolated, library OS instances can be achieved through the pragmatic reuse of networking protocols. Each instance has significantly lower overhead than a full VM bundled with an application: a typical application adds just 16MB of working set and 64MB of disk footprint. We contribute a new ABI below the library OS that enables application mobility. We also show that our library OS can address many of the current uses of hardware virtual machines at a fraction of the overheads. This paper describes the first working prototype of a full commercial OS redesigned as a library OS capable of running significant applications. Our experience shows that the long-promised benefits of the library OS approach—better protection of system integrity and rapid system evolution—are readily obtainable.

1. Introduction

The *library OS* approach to OS construction was championed by several operating system designs in the 1990s [3, 10, 13, 21]. The idea of the library OS is that the entire *personality* of

the OS on which an application depends runs in its address space as a library. An OS personality is the implementation of the OS's application programming interfaces (APIs) and application-visible semantics; the OS services upon which applications are built. Early proponents of the library OS approach argued primarily that it could enable better performance through per-application customization. For example, a disk-I/O bound application with idiosyncratic file access patterns can realize better performance by using a custom file-system storage stack rather than using the default sequential prefetching heuristics.

Like many of its contemporaries, the library OS approach is largely forgotten, a casualty of the rise of the modern virtual machine monitor (VMM) [8]. While most new OS designs of the time—including library OS designs—ran only a handful of custom applications on small research prototypes, VMM systems proliferated because they could run major applications by reusing existing feature-rich operating systems. The performance benefits offered by library OS designs did not overcome the need for legacy compatibility. On the other hand, the need for security and independent system isolation has increased since the 1990s due to the rise of the Internet.

We revisit the library OS approach to OS construction, prioritizing application compatibility, security isolation, and independent system evolution benefits. Our goal is to realize the benefits of a VMM, but with order-of-magnitude lower overheads. This paper shows for the first time that it is possible to build a library OS running major applications from a feature-rich, commercial OS. We describe a Windows 7 library OS, called *Drawbridge*, running the latest commercial releases of a large set of applications, including Microsoft Excel, PowerPoint, Internet Explorer, and IIS. Windows applications running on Drawbridge have access to core Windows features and enhanced APIs including the .NET common language runtime (CLR) and DirectX.

This paper describes a new top-down approach to building library OSes—an approach that prioritizes application compatibility and high-level OS code reuse and avoids low-level management of the underlying hardware by the library OS. Drawbridge demonstrates that a small set of OS abstractions—threads, virtual memory, and I/O streams—are sufficient to host a Windows 7 library OS and a rich set of applications. This small set of abstractions helps simplify protection of system integrity, mobility of applications, and independent evolution of the library OS and the underlying kernel components. Despite being strongly isolated, Drawbridge applications can still share resources, including the screen, keyboard, mouse, and user clipboard, across independent library OS instances through the pragmatic reuse of networking protocols.

As a structuring principle, we identify three categories of services in OS implementations: *hardware services*, *user services*, and *application services*. Hardware services include the OS kernel and device drivers, which abstract and multiplex hardware, along with file systems and TCP/IP networking stack. User services in the OS include the graphical user interface (GUI) shell and desktop, clipboard, search indexers, etc. Application services in the OS include the API implementation; to an application, these comprise the OS personality. Applications services include frameworks, rendering engines, common UI controls, language runtimes, etc. The

application communicates with application services, which in turn communicate with hardware and user services.

We use these service categories to drive the refactoring of Windows into the Drawbridge library OS. Drawbridge packages application services into the *library OS* and leaves user and hardware services in the *host OS* (see Figure 1). It is important to note that the architecture of the components within the library OS is identical to the architecture of those same components within the original full OS (see Figure 2). The library OS communicates with hardware services in the host OS through a narrow *application binary interface* (ABI), which is implemented by a *platform adaptation layer* and a *security monitor*. The library OS communicates with user services in the host OS using the *remote desktop protocol* (RDP) [28] tunneled through the ABI. Each application runs in its own address space with its own copy of the library OS. The security monitor virtualizes host OS resources through its ABI with the library OS and maintains a consistent set of abstractions across varying host OS implementations. For example, the file system seen by an application is virtualized by the security monitor from file systems in the host OS.

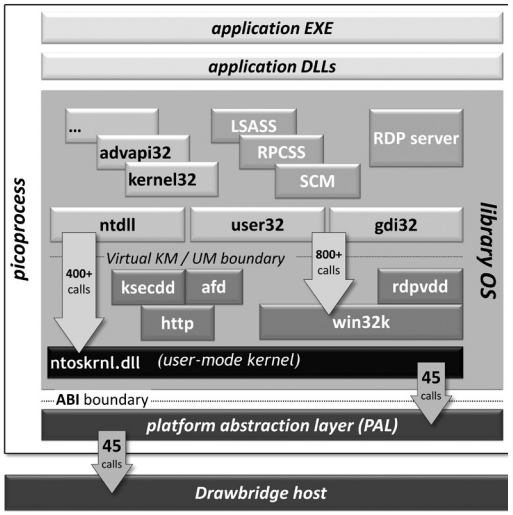


FIGURE 1. Drawbridge Architecture.

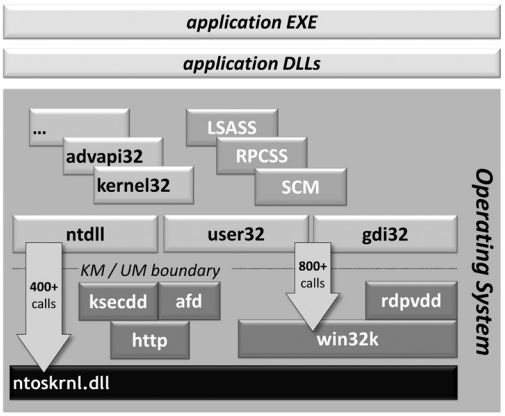


FIGURE 2. Windows 7 OS Architecture.

Previous library OS designs aimed to provide application-customized performance enhancement, and thus exposed low-level hardware abstractions to applications. Cache Kernel, Exokernel, and Nemesis innovated by providing applications with fine-grained, customized control of hardware resources, such as page tables, network packets, and disk blocks [10, 13, 21]. In contrast, Drawbridge's differing goals (security, host independence, and migration) free it to offer higher-level abstractions. These higher-level abstractions make it easier to share underlying host OS resources such as buffer caches, file systems, and networking stacks with the library OS. By making low-level resource management an independent concern from OS personality, each can evolve more aggressively.

One can view a modern VMM as a mechanism for automatically treating a conventional OS as a library OS, but the facility incurs significant overheads. Each isolated application runs in

a different dedicated VM, each of which is managed by a separate OS instance. The OS state in each VM leads to significant storage overheads. For example, a Windows 7 guest OS in a Hyper-V VM consumes 512MB of RAM and 4.8GB of disk. In contrast, Drawbridge refactors the guest OS to extract just those APIs needed by the application; it adds less than 16MB of working set and 64MB of disk.

The Drawbridge approach could significantly impact desktop computing by enabling fine-grain packaging of self-contained applications. The VMM approach allowed the construction of self-contained application packages comprised of an application and OS, with minimal dependencies on the underlying VMM or hardware. VMM-based application packaging enabled a huge shift in server computing through server consolidation and cloud computing [1, 25], all despite huge overheads. We believe that the finer-grained, higher-performance application and OS packages that are now possible with library OSes could precipitate similar shifts in desktop and mobile computing; for example, snapshots of running Drawbridge applications could easily move from device to device and to the cloud because they are so small—a compressed process snapshot of Excel is under 4MB for Drawbridge, versus nearly 150MB for a similar VM snapshot.

In summary, this paper describes the first refactoring of a widely used, monolithic OS into a functionally-rich library OS. It contributes a set of heuristics for refactoring a monolithic kernel into a library as well as a new ABI for separating a library OS from host OS. The benefits of this design include: 1) strong encapsulation of the host OS from the library OS, enabling rapid and independent evolution of each; 2) migration of running state of individual applications across computers; and 3) better protection of system and application integrity (i.e., strongly isolated processes).

The remainder of the paper is structured as follows. Section 2 provides helpful background on the Windows 7 OS architecture as it exists prior to Drawbridge. Section 3 describes our approach for separating the Win32 personality from the rest of Windows to create a library OS. Section 4 describes the security monitor and the ABI it presents to the library OS. Section 5 describes our implementation of the Windows 7 library OS. Section 6 evaluates the scalability, complexity, update surface, and overheads of our approach through examination of the current implementation. Section 7 places Drawbridge into context with related work. In Section 8 we discuss possible impacts of the library OS approach on end-user computing. Finally, we summarize our contributions in Section 9.

2. Windows 7 Architecture

Refactoring a general-purpose, commercial operating system away from its long-evolved roots into a library OS is a significant challenge. This section outlines the key features of the Windows 7 architecture, drawing analogies between elements in Windows and corresponding elements common to such Unix systems as Linux. We draw these analogies to help the reader appreciate that similar challenges exist in other large-scale OSes.

Windows is architecturally divided into *dynamic link libraries* (DLLs) that load into an

application's address space, *services* that run as daemon processes, the *NT kernel*, and *drivers*, which are dynamically loaded kernel-mode components (see Figure 2). DLLs are similar to shared object (.so) files in Unix systems. The NT kernel implements the usual core of a monolithic operating system: resource management, scheduling, I/O services, a hierarchical object namespace, and the *registry*, a key-value store for configuration data. Replaceable kernel-mode components, such as the networking stack and file systems, are loaded as drivers.

In practice, all Windows applications are programmed to the Win32 API. While there is no official count, Win32 is known to include over 100,000 API functions. The API is implemented as a large collection of in-process DLLs. These DLLs access the NT kernel indirectly through a runtime DLL, `ntdll`, which the kernel inserts into every process during creation. `ntdll` implements the dynamic loader for DLLs along with functionality roughly equivalent to the Unix `libc`, including stubs for the 401 functions in the NT system-call table. Commonly used Win32 API DLLs include: `kernel32`, which provides access to kernel operations for process control, threading, virtual memory, registry, and block and character device I/O functionality; `user32` and `gdi32`, which provide the basics of windowing, drawing, and GUI; `ws2_32`, which provides a sockets interface to the networking stack; and `ole32`, which provides access to the Component Object Model (COM) and Object Linking and Embedding (OLE) APIs for constructing multi-component application experiences (e.g., a PowerPoint presentation with an embedded Excel chart). These DLLs correspond to libraries such as `libX11` and the libraries that make up the GNOME or KDE frameworks on Unix systems.

The core of Win32 is implemented in the Windows subsystem. Divided into a kernel-mode component (`win32k`) and a user-mode daemon (`csrss`), the Windows subsystem provides roughly the analogue of an X server, a print server, and audio support. Most of the implementation resides in `win32k`, which implements windowing event message queues, overlapping windows, rasterization, font management and rendering, the mouse and keyboard input event queues, and a shared clipboard. `csrss` coordinates system initialization, shutdown, and application error reporting. Prior to Windows NT 4.0 (1996), functionality now in `win32k` ran in the `csrss` service daemon, but was moved to kernel mode to improve performance and to simplify the implementation of accelerated graphics drivers. User-mode API DLLs access `win32k` indirectly through stubs in `user32` and `gdi32`; the stubs trap into a secondary system-call table of 827 functions from `win32k`.

In addition to the Windows subsystem, the implementation of the Win32 API depends on multiple service daemons. `smss` performs a role similar to Unix `init`, handling startup and shutdown. `wininit` creates read-only shared data structures, which are subsequently mapped into most processes and used by `win32k` to cache such common objects as default fonts, internationalization tables, and cursors. `wininit` also launches the components of the user's desktop upon login, the analogue of an X session manager. `rpcss` implements shared services for high-level inter-process communication, including COM and OLE, similar to D-Bus [24] in Linux. `explorer`, the Windows GUI shell, launches programs and provides shared services for drag-and-drop, "open" and "save" dialogs, and file preview. `dwm` implements the Windows 7 compositing window manager.

3. Approach

To maximize application compatibility while minimizing dependencies outside the library OS, we refactored Windows 7 by applying four high-level heuristics: inclusion of API DLLs based on their usage in a representative set of applications, reuse of virtualized host OS resources, resolution of dependencies through inclusion or alternative implementations, and device driver emulation. One insight we applied repeatedly in our work is the recognition that much of the code in an OS kernel and major OS subsystems is not relevant in the context of a library OS. For example, much OS code ensures security and consistency when sharing resources—either physical or virtual—between multiple applications and multiple users. In a library OS, these concerns are greatly diminished as library OS state is not shared by multiple applications or users.

Our first heuristic was to identify the API DLLs required by a representative set of applications. Those DLLs account for over 14,000 functions of the Win32 API. We used static analysis on the application binaries to roughly approximate the required set of API DLLs, and then refined the set with dynamic instrumentation by monitoring DLL load operations issued during test runs. In our experience, static analysis alone is either insufficient—DLLs can be loaded without static stubs through calls to `LoadLibrary` (equivalent to `dlopen` on Unix) with a dynamically generated string—or overly conservative—including delay-bound DLLs loaded only for specific OS versions.

Second, for kernel-mode dependencies, we implemented an NT kernel emulation layer at the bottom of the library OS. This emulation layer is quite thin, as many complex parts of a kernel—like threading, virtual memory, file system, and networking—are provided by the host OS through the security monitor. The security monitor virtualizes host resources according to a well-defined high-level ABI independent of host OS version. Other parts of the library OS are simpler because multi-user multiplexing is no longer required; for this reason the Drawbridge registry implementation is 1/50th the lines of code of the Windows 7 equivalent. While our emulation provides most of the interfaces of the NT kernel, many calls return failure, including all requests to access or modify other processes, and almost all `ioctl` requests. In theory the "holes" in our emulation of the NT kernel could cause an application to fail unexpectedly due to an unimplemented corner case. In practice, we find that most applications either don't use the unimplemented interfaces, or respond gracefully when a rarely-used API returns a failure result.

Third, for dependencies on service daemons and the Windows subsystem, we either moved code into the library OS, or altered the API DLL to remove the dependency. As a rule of thumb, we included code where most of the service was relevant when running a single application, and replaced code where it was needlessly complicated by the security or consistency demands of supporting multiple applications and/or multiple users. For example, we included almost all of win32k and rpcss, as these services provide core functionality for applications. By contrast, we wrote custom library OS code to replace `csrss`, `smss`, and `wininit`, which primarily aid cross-application sharing of state.

Fourth, for console and human interface device dependencies, we provide emulated device drivers. We emulate the keyboard and mouse drivers required by the Windows subsystem with stub drivers that provide simple input queues, and the display driver with a stub driver that draws to an in-process frame buffer. I/O from the emulated devices is tunneled to the desktop and the user through stateless RDP connections. Our implementation of RDP reuses code from the Windows 7 kernel-mode RDP server but not the RDP device drivers, which are replaced by simpler stubs.

4. Security Monitor

The library OS interacts with the host OS through the Drawbridge ABI, which is implemented by the security monitor. The Drawbridge ABI is designed to provide a small set of functions with well-defined semantics easily supported across a wide range of host OS implementations. The ABI's design enables the host OS to expose virtualized resources to the library OS with minimal duplication of effort. We describe here first the ABI and then the implementation of the security monitor.

In providing the ABI, the security monitor enforces a set of external policies governing the host OS resources available to the application. Inspired by previous work in Singularity [33], we encode policy in manifest files associated with the application. The manifest whitelists the host OS resources that an application may access, identified by URI path. We also use the manifest as a convenient place to store per-application configuration settings.

ABI Description

The ABI includes three calls to allocate, free, and modify the permission bits on page-based virtual memory. Permissions include read, write, execute, and guard. Memory regions can be unallocated, reserved, or backed by committed memory:

```
VOID *DkVirtualMemoryAlloc(Addr, Size, AllocType, Prot);
DkVirtualMemoryFree(Addr, Size, FreeType);
DkVirtualMemoryProtect(Addr, Size, Prot);
```

The ABI supports multithreading through five calls to create, sleep, yield the scheduler quantum for, resume execution of, and terminate threads, as well as seven calls to create, signal, and block on synchronization objects:

```
DKHANDLE DkThreadCreate(Addr, Param, Flags);
DkThreadDelayExecution(Duration);
DkThreadYieldExecution();
DkThreadResume(ThreadHandle);
DkThreadExit();
DKHANDLE DkSemaphoreCreate(InitialCount, MaxCount);
DKHANDLE DkNotificationEventCreate(InitialState);
DKHANDLE DkSynchronizationEventCreate(InitialState);
DkSemaphoreRelease(SemaphoreHandle, ReleaseCount);
BOOL DkEventSet(EventHandle);
DkEventClear(EventHandle);
ULONG DkObjectsWaitAny(Count, HandleArray, Timeout);
```

The primary I/O mechanism in Drawbridge is an I/O stream. I/O streams are byte streams that may be memory-mapped or sequentially accessed. Streams are named by URIs. The stream ABI includes nine calls to open, read, write, map, unmap, truncate, flush, delete and wait for

I/O streams and three calls to access metadata about an I/O stream. The ABI purposefully does not provide an `ioctl` call. Supported URI schemes include `file:`, `pipe:`, `http:`, `https:`, `tcp:`, `udp:`, `pipe.srv:`, `http.srv`, `tcp.srv:`, and `udp.srv:`. The latter four schemes are used to open inbound I/O streams for server applications:

```
DKHANDLE DkStreamOpen(URI, AccessMode, ShareFlags, CreateFlags, Options);
ULONG DkStreamRead(StreamHandle, Offset, Size, Buffer);
ULONG DkStreamWrite(StreamHandle, Offset, Size, Buffer);
DkStreamMap(StreamHandle, Addr, ProtFlags, Offset, Size);
DkStreamUnmap(Addr);
DkStreamSetLength(StreamHandle, Length);
DkStreamFlush(StreamHandle);
DkStreamDelete(StreamHandle);
DkStreamWaitForClient(StreamHandle);
DkStreamGetName(StreamHandle, Flags, Buffer, Size);
DkStreamAttributesQuery(URI, DK_STREAM_ATTRIBUTES *Attr);
DkStreamAttributesQueryByHandle(StreamHandle, DK_STREAM_ATTRIBUTES *Attr);
```

The ABI includes one call to create a child process and one call to terminate the running process. A child process does not inherit any objects or memory from its parent process and the parent process may not modify the execution of its children. A parent can wait for a child to exit using its handle. Parent and child may communicate through I/O streams provided by the parent to the child at creation:

```
DKHANDLE DkProcessCreate(URI, Args, DKHANDLE *FirstThread);
DkProcessExit(ExitCode);
```

Finally, the ABI includes seven assorted calls to get wall clock time, generate cryptographically-strong random bits, flush portions of instruction caches, increment and decrement the reference counts on objects shared between threads, and to coordinate threads with the security monitor during process serialization:

```
LONG64 DkSystemTimeQuery();
DkRandomBitsRead(Buffer, Size);
DkInstructionCacheFlush(Addr, Size);
DkObjectReference(Handle);
DkObjectClose(Handle);
DkObjectsCheckpoint();
DkObjectsReload();
```

We believe the brevity of the Drawbridge ABI enables tractable coding-time and run-time review of its isolation boundary. Our largest implementation of the ABI in a security monitor is 17KLoC.

We recognize that the Drawbridge ABI could be smaller. While size matters, we have in a few cases opted for exposing a slightly larger set of abstractions than was strictly necessary in order to ease porting of Windows code into the library OS. Our experience is that a slightly larger ABI (say with a dozen more calls) makes it easier to port existing code and makes the ported code easier to maintain. For example, instead of exposing semaphores, notification events, and synchronization events, we could have exposed only a single synchronization primitive. In fact, the initial version of the ABI had just 19 calls, with synchronization objects and I/O streams being coalesced into a single pipe abstraction; for example, lock release was implemented in the library OS as a one-byte send on a pipe and acquire as a one-byte write. The ABI was smaller, but reasoning about library OS implementation was more torturous.

The slightly larger ABI allows the security monitor to implement virtualized resources with resources that the host OS can more efficiently support.

Implementation

The Drawbridge ABI is implemented through two components: the security monitor, `dkmon`, and the platform adaptation layer, `dkpal`. The primary job of `dkmon` is to virtualize host OS resources into the application while maintaining the security isolation boundary between the library OS and the host OS. Although implementations of `dkmon` and `dkpal` vary across different host systems, these components are responsible for maintaining strict compatibility with the ABI specification. We currently have implementations of `dkmon` that run Drawbridge applications as processes on Windows 7 and Windows Server 2008 R2, on MinWin [36] built from Windows 7, and on a pre-release of the next version of Windows. We also have a version of `dkmon` that runs Drawbridge applications in ring 0 in a raw Hyper-V VM partition, while relying on I/O streams served from a Windows Server 2008 R2 host. Applications using the Drawbridge library OS run identically on all of these platforms.

A Drawbridge process accesses the ABI by calling `dkpal`. We have three implementations of `dkpal`: the first, which requires no changes to the host OS kernel, and uses four host OS calls to issue requests over an anonymous named pipe to `dkmon`; the second, which replaces the NT system-call service table on a per-process basis using techniques developed for Xax [11]; and the third, which makes Hyper-V hypercalls. `dkmon` services ABI requests by modifying the address space and host OS handle table of the calling process with standard Windows cross-process manipulation APIs (`ReadProcessMemory`, `VirtualAllocEx`, `VirtualFreeEx`, and `DuplicateHandle`). As an optimization, `dkpal` implements a few simple ABI calls (e.g., blocking wait, thread yield) by directly invoking compatible, host OS system calls; this is safe, as the Drawbridge process cannot create host OS handles. `dkpal` currently calls 15 distinct host OS system calls. Eventually, we expect to move the data paths `dkmon` into the host kernel to avoid the cost of a complete address space change to service some ABI calls, and to harden the boundary around Drawbridge processes; this will require an update to `dkpal`, but no changes in the Drawbridge library OS.

`dkmon` uses host NT threads and synchronization objects—semaphores, notification events, and synchronization events—to implement the scheduling objects exposed through the ABI. As a result, Drawbridge threads reside in the host kernel's scheduling queues and avoid unnecessary scheduling overheads.

I/O streams are used by the library OS to implement such higher-level abstractions as files, sockets, and pipes. The security monitor filters access to I/O streams by URI based on a manifest policy; where access is allowed, it directs I/O to the mapped resources. This indirection enables run-time configuration of the application's virtual environment, and prevents applications from inadvertently or maliciously accessing protected resources within the host system's file namespace. Unless overridden, the monitor's default policy only allows a Drawbridge process to access files within the same host directory as its application image.

As an example, our library OS leverages I/O streams to emulate NT file objects and named pipe objects, as well as a proxied interface to networking sockets. In the latter case, the library OS includes a minimal version of `ws2_32` that use I/O streams identified by `tcp:` and `udp:` URIs. The security monitor backs these streams with sockets provided by the host system's `ws2_32`. The current approach was taken as an implementation expediency; Drawbridge could provide better isolation by offering an IP-packet stream interface and moving the implementation of TCP and UDP into the library OS.

5. Windows Library OS

Refactoring Windows 7 into a library OS constituted the largest portion of the work to realize Drawbridge. The main challenges included: orchestrating process bootstrap with minimal changes to existing components, emulating host kernel interfaces in user mode on the Drawbridge ABI, porting system-wide application services to run in-process, and enabling process serialization and deserialization. We believe that these challenges would be typical of the refactoring effort required to convert any complex, modern operating system into a library OS for desktop applications, and that our solutions would apply elsewhere. However, we also recognize that our task was made significantly easier because most Windows applications seldom use child processes and the Windows API has no `fork` primitive.

OS Library Bootstrap

Bootstrapping is a challenging task in any OS, balancing the demands of efficiency and good engineering. For example, should one first initialize the lock manager, which requires memory for storage, or the memory manager, which requires locks for synchronization? Some of these complexities remain for the library OS, which aggregates per-process code and state with formerly system-wide OS code and state.

Process Bootstrap. To create a Drawbridge process, `dkmon` uses the host OS's native facilities to create a suspended process containing a bootstrap loader (`dkinit`). Every NT process is created with the `ntdll` library mapped copy-on-write, because the kernel uses fixed offsets in the library as up-call entry points for exceptions. Before allowing a new process to execute, `dkmon` maps its own `dkntdll` library into the new process's address space and overwrites upcall entry points in the host-provided `ntdll` with jumps to `dkntdll`, eviscerating `ntdll` to a jump table and replacing it as the dynamic loader. `dkmon` writes a parameter block into the new process's address space to communicate initialization parameters, such as a reference to the pipe to be used for communication with `dkmon`. `dkmon` then resumes the suspended process, with execution starting in `ntdll` and immediately jumping to `dkntdll`, which sets up initial library linkage (to itself) and transfers control to `dkinit`. `dkinit` invokes `dkntdll` to initialize the `win32k` library (described next) and to load the application binary and its imported libraries. When loading is complete, `dkinit` jumps to the application's entry point.

Win32k Bootstrap. Converting `win32k` from a kernel subsystem to a user-mode library required reformulating its complicated, multi-process initialization sequence. In standard Windows, first, the single, system-wide instance of `win32k` is initialized in kernel mode.

Second, wininit initiates the preloading of win32k's caches with shared public objects such as fonts and bitmaps. Because win32k makes upcalls to the user32 and gdi32 user-mode libraries to load an object into its cache, these libraries must be loaded before filling the cache. Third, when a normal user process starts, it loads its own copies of user32 and gdi32, which connect to win32k and provide GUI services.

We considered refactoring the initialization of win32k, but rejected that idea as it required a deeper fork of the source code than desired and prevented sharing of code between full Windows and Drawbridge. Instead, we opted for an approach that simulates the full win32k bootstrapping sequence within a single process. We exported entry points from win32k, user32, and gdi32, which are called by dkinit for each of the boot steps.

Much of the effort described here can be viewed as unwinding the complexity of the multi-server architecture of Windows into an in-process library OS. On a full Windows system, csrss creates a read-only, shared-memory segment to share cached bitmaps and fonts, which is replaced in the library OS with heap allocated objects, since all components that access it now share the same address space and protection domain. The upcalls that torture the win32k initialization sequence were needed only because a shared, trusted win32k must avoid being confused into loading one principal's objects on behalf of another; in Drawbridge, that responsibility is removed, and the corresponding complexity can be reduced. Likewise, we removed many other access checks over shared state from win32k, and eliminated the Windows logon session abstraction, wininit, and csrss.

Emulating NT Kernel Interfaces

To support binary compatibility with existing Windows 7 API DLLs, we provide user-mode implementations of approximately 150 NT kernel system calls. The majority of these functions are stubs that either trivially wrap the Drawbridge ABI (e.g., virtual memory allocation, thread creation, etc.), return static data, or always return an error (i.e., STATUS_NOT_IMPLEMENTED).

The remaining system calls produce higher-level NT abstractions, such as files, locks, and timers, built entirely inside the library OS using Drawbridge primitives. The most challenging part of this work was building compatible semantics for the NT I/O model, including synchronous and asynchronous I/O, "waitable" file handles, completion ports, and asynchronous procedure calls.

Additional NT emulation calls were required to move win32k from kernel mode to user mode. The interfaces provided to kernel-mode libraries by the NT kernel are largely disjoint from those provided to user-mode libraries; we added roughly 6.3KLoC to emulate these kernel-mode calls.

Shared System Services

Windows applications depend on shared system services, accessible either through system calls or IPCs to trusted service daemons. These include services such as the Windows registry and OLE. In order to confine the state and dependencies of Drawbridge processes, our library OS implements the functionality of several such services using two design patterns: providing

simple alternate implementations of service functionality, and hosting extant library code in-process.

Alternative Implementations. Many Windows system services are backed by complex and robust implementations, tuned and hardened for a wide variety of use cases. For a few such services like the registry, it proved more practical to reimplement the services' advertised interfaces within the library OS rather than port their existing implementations.

The *registry* is a system-wide, hierarchical, key-value store, accessible to Windows processes through system calls. The traditional Windows registry is implemented in kernel mode with 61 KLoC; its complexity is required to implement fine-grained access control and locking as well as transactional semantics. While we might have replicated code for the kernel registry into the library OS, we found that registry code is inseparably connected to code for the kernel object namespace and the kernel scheduler. The Drawbridge library OS instead includes a private, in-process reimplementation of the registry, significantly simpler than the shared kernel registry. Drawbridge's NT emulation layer supplies a simple 1.3 KLoC interface to this implementation, with coarse locking and no support for transactions.

Importing Implementations. In several cases, we encountered shared Windows services whose implementations could be ported largely intact. As an example, Drawbridge's support for COM required functionality traditionally provided by `rpcss`. Supporting COM is an absolute requirement for rich applications in Windows; a significant number of desktop- and server-class applications are formed by composing multiple COM objects through the OLE protocol. With one exception (Reversi) each of our test applications uses OLE.

Refactoring COM followed the same basic pattern used with other system services: shared, out-of-process components were ported to run privately in-process and bound directly to application-side libraries. For example, a key component of OLE is the running object table (ROT), which provides inter- and intra-process name resolution for COM objects. While only one instance of the ROT is maintained per system within `rpcss`, in Drawbridge it runs directly within the application process and only manages local objects. Fewer than 500 lines out of 318 KLoC were changed in the COM runtime (`ole32`) to make these changes.

Process Serialization

The volatile state associated with a traditional Windows process is distributed across the NT kernel and kernel-mode drivers; shared, out-of-process, user-mode service daemons; and the process's own address space. Serializing the running state of a Windows process would require the careful cooperation of each of these components, and significant changes to the OS. However, with Drawbridge process isolation, a process's transient state is either confined to pages in its address space or can be reconstructed from data maintained within its address space.

Due to careful design of the ABI and library OS, serializing a Drawbridge process is relatively simple. Running `win32k` as a user-mode library vastly reduced the amount of kernel state associated with the process. The remaining out-of-process state consists of resources

managed by the host OS, such as files and synchronization objects. To account for this, our implementation of the ABI inserts indirection between Drawbridge system objects and host NT system objects. This distinction enables the library OS to easily unbind and rebind these objects at deserialization time. Because the metadata for host kernel objects are stored within a process's address space, serialization only requires quiescing the threads and serializing the contents of the address space. The thread contexts need not be serialized, as the active register contents are stored on their stacks during quiescence. The application serializes itself with no involvement from the host, beyond the I/O stream to which the serialized state is saved.

In order to quiesce the threads within a Drawbridge process, the security monitor signals a notification event to indicate a serialization request. Threads blocked on Drawbridge ABIs are woken via the notification event, outstanding I/O requests are completed, and other threads are interrupted with an exception. Once notified of the pending serialization, all threads, except one, yield indefinitely. The final thread begins serializing the process to a monitor-provided I/O stream, recording virtual memory bookkeeping information and the contents of the process's address space. Files on which the process depends must be migrated with the application or accessed through a distributed file system. Network sockets are terminated on migration causing applications to reestablish their network connections. In a world of migrating laptops, we find that applications are robust to network interruptions. After serialization is complete, the yielding threads are woken and the process continues normal execution.

Reconstructing a Drawbridge process from its serialized state requires adjustments to its initialization sequence. Instead of loading the application binary, serialized virtual memory data are used to restore the contents of the process's address space, including heap and stack memory as well as memory-mapped I/O streams. Code in `dkpal` rebinds host system objects to ABI objects without involvement from the library OS or the application. After recreating the process, the threads quiesced during serialization are unblocked and continue normal execution.

Limitations

The present Drawbridge system is a research prototype; it is far from a production system. While Drawbridge supports over 14,000 Win32 API functions, this is a fraction of the total Win32 API. At the time of writing, Microsoft has no plans to productize any of the concepts prototyped in Drawbridge.

The two holes in the current implementation are support for printing and support for multi-process applications that communicate through shared state. The challenge of printing is that most Windows printer drivers actually load in-process with the application. As our design requires leaving hardware drivers within the host OS, they can't be loaded into the library OS without creating undesirable dependencies. A possible solution is to reuse the approach of RDP 7.0, which employs a universal application printer driver that prints to a common format, XPS, and then asks the RDP client's printer to print the XPS.

Solving the problem of multi-process applications is much harder, particularly for applications that communicate through shared state in `win32k`, as is done in many OLE scenarios. For

example, Microsoft Outlook can be configured to use Microsoft Word as a text editor with the shared state passing through `win32k` message queues. We have considered, but not implemented, two possible designs. One is to load multiple applications into a single address space. Another is to run `win32k` in a separate user-mode server process that can be shared by multiple applications in the same isolation container. The latter approach follows a pattern commonly used by microkernel designs.

A third weakness in the present implementation is that code paths in many Win32 APIs ultimately lead to NT APIs that are not implemented by our emulation layer. We have ported the most frequently used subset of a very large API space. As we have run new applications, the number of additions to the library OS has diminished, but not completely disappeared. For example, after Excel ran, getting PowerPoint to run on Drawbridge took only two days: one to fix a bug we had introduced in `win32k`, and one to implement an additional API in the NT emulation layer.

Finally, there are classes of applications that, by design, will probably never run in Drawbridge. For example, administration tools, that manipulate the host OS, or development tools such as debuggers, which need unfiltered access to the environment, cannot be run without creating hard dependencies on the host OS.

6. Experiments

This section evaluates the theses of the Drawbridge project: that a library OS can run rich desktop applications, that such refactoring is feasible, that it is suitable for isolating large numbers of applications in a single computer, that it protects the integrity of the host OS at least as well as a VMM, that it provides greater mobility for running applications, and that it enables independent evolution of host OS from library OS. We also measure the servicing implications of the library OS.

All data were collected on an HP z800 Workstation with dual 2.4GHz Intel Xeon E5530 Quad-Core CPUs with hyper-threads disabled, 16GB of RAM, and dual 10,000 RPM hard drives. All Windows experiments use 64-bit Windows 7, Ultimate Edition, with the page file disabled. All Hyper-V experiments run on 64-bit Windows Server 2008 R2, Enterprise Edition, which shares the same code base as Windows 7. Drawbridge and all applications are 64-bit binaries. Windows 7 and Hyper-V were tuned for maximum scalability according to published best practices [27].

For most experiments, we present results for three applications: Excel, a canonical desktop application; Internet Explorer, a web browser and network client application; and IIS, a canonical server application. Unless stated otherwise: Excel experiments used small (11KB), large (20MB), and huge (100MB) spreadsheets; Internet Explorer rendered `research.microsoft.com`, and IIS was serving up the default Visual Studio 2010 ASP.NET application using CLR 4.0.

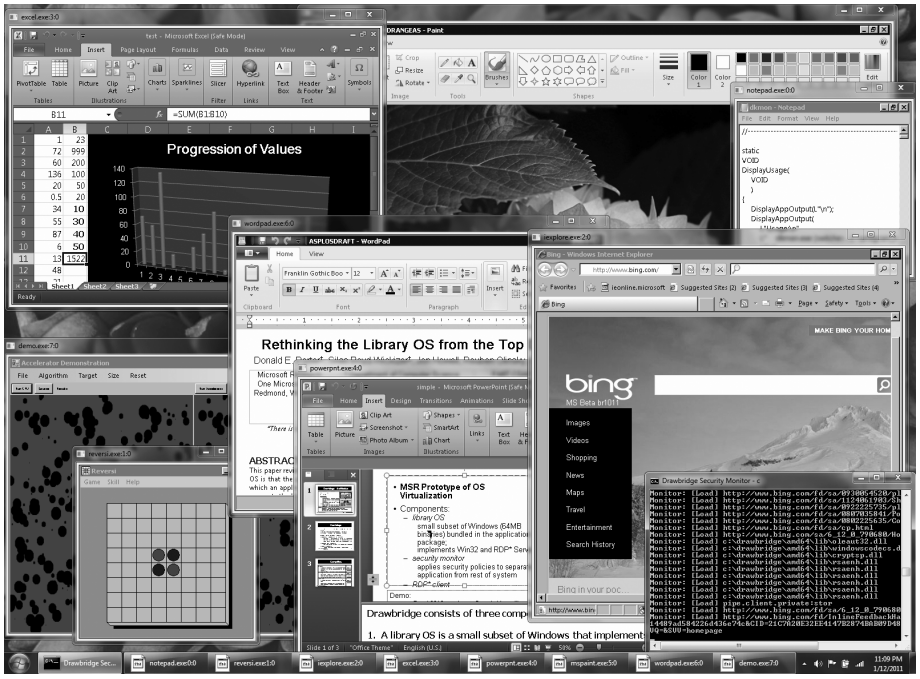


FIGURE 3. Screenshot of Drawbridge Applications: Clockwise from the top-left: Excel, Paint, Notepad, dkmon, Internet Explorer, WordPad, PowerPoint, Reversi, and a CLR Demo.

Running Applications

The primary hypothesis of Drawbridge is that a legacy OS can be refactored into a resource-efficient library OS that supports a large class of rich, desktop applications. Figure 3 shows a sample screenshot with output from `dkmon` and seven applications running in Drawbridge: Microsoft Excel 2010, Windows 7 Paint, Notepad, Internet Explorer 8, PowerPoint 2010, Windows 7 WordPad, a CLR demo application, and Reversi, the first game to run on Windows. Other applications we have tested include IIS 7.5, DirectX 11 demos, and a number of in-house applications. With the exception of disabling software licensing in Excel and PowerPoint, all application binaries are unmodified.

To run on Drawbridge, applications were "sequenced", by running their setup programs on a desktop copy of Windows 7, and capturing the file-system and registry changes made by the setup program into a Drawbridge package. Tools for collecting file-system and registry changes are well understood and deployed with products such as Microsoft Application Virtualization (App-V) and VMWare ThinApp [37].

Cost of Refactoring

Our hypothesis maintains that the refactoring task is tractable. Of the 93 binaries containing executable code in the Drawbridge library OS, 62 come directly from Windows 7 with no

Binary	#if's	Changed LoC	Total LoC	Size (KB)
advapi32.dll	18	720	61,975	655
dxapi.dll	88	294	3,225	18
dxg.dll	110	449	12,706	85
dxgi.dll	4	20	46,347	616
dxgkrnl.dll	1	8	518	9
gdi32.dll	9	70	43,711	374
kernel32.dll	37	262	153,905	944
kernelbase.dll	21	188	40,734	312
msvcrt.dll	1	5	70,201	590
ntdll.dll	83	4,274	148,327	1,484
ole32.dll	188	915	196,706	1,907
oleaut32.dll	5	34	84,331	763
rdp4vs.dll	5	647	50,312	261
rdpclip.dll	18	111	21,306	128
rdpvdd.dll	1	666	4,312	30
rpcrt4.dll	5	34	135,812	959
user32.dll	68	497	60,161	935
win32k.dll	685	5,341	343,082	2,845
winhttp.dll	15	1,146	6,225	40
New Implementations: clbcatq.dll, ddraw.dll, dwmapi.dll, iphlpapi.dll, msi.dll, netapi32.dll, sechost.dll, secur32.dll, wininet.dll, winspool.drv, ws2_32.dll, wtsapi32.dll			25,984	251
Unchanged: atl.dll, comctl32.dll, comdlg32.dll, comsvcs.dll, crypt32.dll, cryptbase.dll, cryptsp.dll, d3d10.dll, d3d10_1.dll, d3d10_1core.dll, d3d10core.dll, d3d10level9.dll, d3d10ref.dll, d3d10sdklayers.dll, d3d10warp.dll, d3d11.dll, d3d11ref.dll, d3d11sdklayers.dll, d3d8thk.dll, d3d9.dll, d3dcompiler_42.dll, d3dx10_42.dll, d3dx11_42.dll, d3dx9_42.dll, dbghelp.dll, ddrawex.dll, dui70.dll, duser.dll, explorerframe.dll, fms.dll, gdiplus.dll, iertutil.dll, imagehlp.dll, mfc42u.dll, mlang.dll, msasn1.dll, msftedit.dll, msls31.dll, mswsock.dll, odbc32.dll, oleacc.dll, profapi.dll, propsys.dll, psapi.dll, rdpd3d.dll, rgb9rast.dll, rsaenh.dll, shell32.dll, shfolder.dll, shlwapi.dll, sspicli.dll, uiribbon.dll, urlmon.dll, userenv.dll, usp10.dll, uxtheme.dll, version.dll, windowscodecs.dll, winmm.dll, wintrust.dll, wsock32.dll, xmllite.dll			3,995,244	57,912
Totals	**1,362**	**15,681**	**5,505,124**	**71,118**

Table 1. Summary of changes to Windows 7 executable binaries to produce the Drawbridge library OS.

modifications, 12 are alternative implementations (mostly machine generated stubs), and 19 contain modifications. The changes are generally quite small when compared with the number of lines of code that remain unchanged (see Table 1). The most significant changes were in: `gdi32`, `ntdll`, and `user32`, which no longer trap, but now directly call either `win32k` or the NT emulation layer; `dxapi`, `dxg`, and `win32k`, which were kernel-mode drivers now modified to run as user-mode DLLs; and `ole32` and `wininet`, in which we removed dependencies on external services. Supporting our hypothesis, repurposing 5.6 MLoC of Windows 7 into a library OS required less than 16 KLoC of changes (0.3% of code base) and 36 KLoC of new code; the entire project was completed in fewer than two person-years.

Overheads

To validate the hypothesis that Drawbridge has modest resource overheads, we measure committed memory and start times for applications running natively on Windows, on Drawbridge, and on Hyper-V. Figure 4 shows the amount of committed memory required for each OS configuration and application. All VMs were configured with 512MB of RAM, except Excel 100MB test, which has 1024MB to avoid excessive paging. With 512MB, the startup time is over 260 seconds. The additional committed memory of Drawbridge is basically the

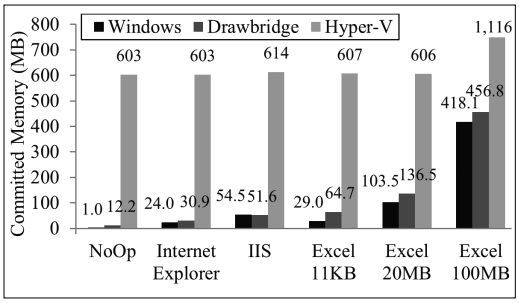


FIGURE 4. Memory per application (including memory used by library OS or guest OS).

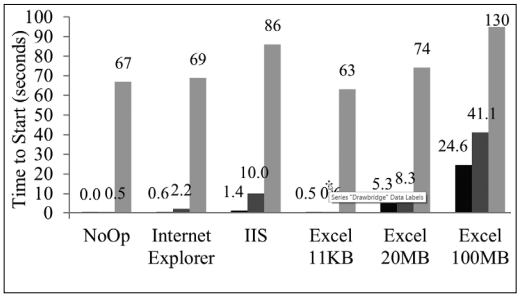


FIGURE 5. Time to start application (including time to start library OS or guest OS).

cost of each application running a private copy of `win32k`, including its fonts and graphic object caches. Running the application in a VMM, on the other hand, incurs the full cost of running a complete guest OS. Similarly, startup times for a full guest OS are much higher than for a library OS (see Figure 5). Ongoing execution overheads are only slightly higher with Hyper-V (for our applications typically less than 1%).

Figure 6 shows the aggregate effect of memory for Excel. While Hyper-V can host only 23 isolated copies of Excel (each with a 20MB spreadsheet loaded), Drawbridge can host 104 instances on the same hardware, compared to 142 instances if Excel is run as a native Windows application with no isolation. Drawbridge is sufficiently efficient that every application can be run in its own isolation container. For Internet Explorer, we found that Drawbridge can host 527 instances, Windows can host 138, and Hyper-V can host 22 (see Figure 7). In the

case of Windows, Internet Explorer exhausts the hard limit on GDI handles per logon session in `win32k`, not physical memory. Drawbridge does not have this limit as each library OS has its own private `win32k`. Finally, for IIS, we found that Drawbridge can host 287 instances, Windows can host 266 and Hyper-V can host 21 (see Figure 8).

In terms of disk footprint, the Drawbridge library OS is 64MB (or 83MB with DirectX 11 support) compared to 4.2GB for a guest OS copy of Windows 7. If desired, the library OS can be further reduced to meet the needs of a single application; a copy of the library OS reduced to its minimum for running Reversi is under 16MB.

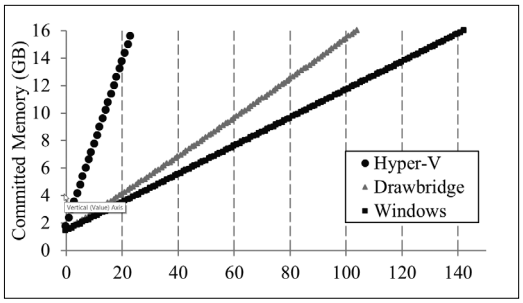


FIGURE 6.Memory committed for increasing copies of Excel, each with a 20MB spreadsheet loaded.

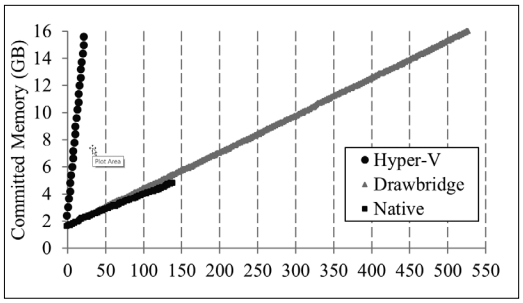


FIGURE 7. Memory committed for increasing copies of Internet Explorer rendering research.microsoft.com.

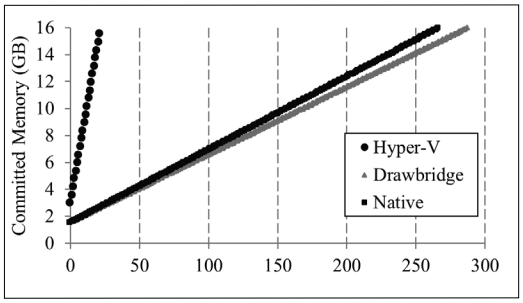


FIGURE 8. Memory committed for increasing copies of IIS.

Protecting System Integrity

To illustrate that Drawbridge applications are isolated and cannot harm the host OS, we performed two simple experiments and conducted a case study based on a recently-published Internet Explorer exploit [20]. First, we ran a toy malware program that deletes every registry key, which would normally leave a system unusable, and found that only the malware was affected. Second, we wrote a simple key logger using the `SetWindowsHookEx` function. When run outside of Drawbridge, the key logger collected every keystroke issued. Inside Drawbridge, the key logger was unable to obtain keystrokes in other applications.

Keetch [20] documents a set of attack vectors and attack patterns to exploit possible weaknesses in the protected mode of Internet Explorer. Protected mode runs a copy of Internet Explorer in a process with restricted OS permissions. Among other restrictions, in protected mode, Internet Explorer should not be able to write to persistent storage. The idea of protected mode is that even if an attacker is able to run exploit code within Internet Explorer, the code cannot escape the low privilege process and therefore no permanent harm can be done to the system. Keetch identifies five attack vectors for exploit code to escape protected mode: name squatting on the NT kernel object namespace,

leaked or duplicated handles, objects deliberately shared between low- and fully-privileged processes, clipboard content spoofing, and spoofing of a web server in the local-domain trusted zone through the loopback interface.

Application	Drawbridge Snapshot	Hyper-V Snapshot
Excel: 11KB file opened	3.1 MB	148.6 MB
Excel: 20MB file opened	21.2 MB	155.8 MB
Excel: 100MB file opened	86.9 MB	313.2 MB
Internet Explorer	3.9 MB	184.8 MB
IIS: no site pre-loaded	1.1 MB	193.4 MB

TABLE 2. Comparison of compressed memory snapshots.

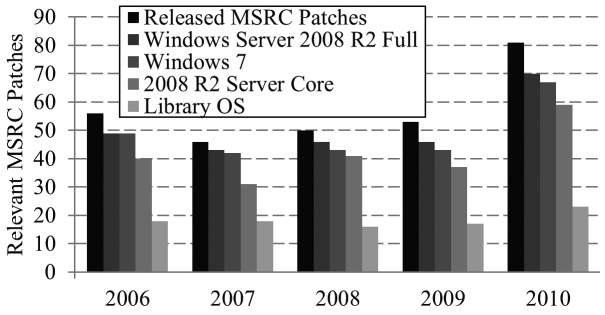


FIGURE 9. Security patches affecting binaries in Windows Server 2008 R2, Windows 7, Windows Server 2008 R2 Server Core, and Drawbridge.

Drawbridge mitigates all five of the attack vectors identified by Keetch. Each application runs with its own library OS that doesn't share an emulated kernel namespace, doesn't share handles, and doesn't share kernel objects. Drawbridge applications can access the clipboard and act as network servers, but only if the ability has been whitelisted for the application. Internet Explorer on Drawbridge is not permitted to act as a web server, thus preventing network spoofing. Access to the clipboard in Drawbridge is set by RDP policy and can also be prevented. By comparison, Windows 7 running in a VMM doesn't share its kernel namespace, doesn't share handles, and doesn't share kernel objects. Furthermore, like Drawbridge, a VMM guest OS can access the desktop clipboard and the local network only if permitted.

Migrating Applications

One benefit of Drawbridge's ABI is that it enables the migration of running applications between computers and across OS reboots. For most Windows applications, the only alternative to Drawbridge is to use a VMM to migrate the application and its full operating system. To compare the cost of migration of an application using Drawbridge versus a Windows 7 guest VM using Hyper-V, we ran five application scenarios on each system, triggered a memory snapshot, and then compressed the snapshot to determine the smallest image. We disabled the page file in Windows 7 on Hyper-V to avoid including the additional size of a page file in the snapshot.

As Table 2 shows, Drawbridge snapshots are significantly smaller, because it doesn't serialize the OS with the application. In practice, we find that application snapshots for Drawbridge are typically in the 3-4MB range; roughly the size of an MP3, they are easily moved over network connections. Application serialization and deserialization generally takes less than 1 second on Drawbridge, and more than 10 seconds for Hyper-V.

Evolving the OS Kernel

By factoring the OS into three components (high-level user shell, library OS, and low-level kernel), Drawbridge enables their independent evolution. To validate this assertion we wrote a version of `dkpal` designed to run as a Hyper-V partition in ring-0, replacing the low-level Windows 7 kernel implementation.

`hvdkpal` is essentially a small OS kernel that implements the Drawbridge ABI described in Section 4. `hvdkpal` implements threading and preemptive scheduling by multiplexing Hyper-V virtual CPUs. `hvdkpal` includes an implementation of futexes [14], which it uses to provide Drawbridge events and semaphores. Since `hvdkpal` runs in ring-0, it is able to manipulate page tables directly in order to manage virtual memory. `hvdkpal` does not implement a file system or networking stack; to support I/O streams, it forwards I/O-related ABI calls to the host Windows Server 2008 R2 kernel running in Hyper-V's root partition. The `hvdkpal` implementation is about 6,000 lines of C.

Despite the large differences between the `hvdkpal` kernel and what the Windows kernel applications expect, `hvdkpal` runs our entire suite of unmodified Windows applications with an unmodified library OS and an unmodified high-level user shell. The applications are oblivious to the fact that they are running in ring-0.

We also ran our set of unmodified applications and library OS on a MinWin [36] build of Windows. MinWin is a minimal build of Windows with less than 48MB of binaries; it consists of the NT kernel, the NTFS file system, TCP/IP stack, and the storage and network drivers for a specific computer. It does not include a shell. For our experiments, we connected the RDP client directly to a graphics frame buffer driver. One of our colleagues has begun to experiment with Windows 7 applications on the Barrelfish OS [6] using the Drawbridge ABI to explore extreme hardware configurations. Our experiments indicate that Drawbridge's approach to factoring OS components enables independent evolution of the low-level kernel from the library OS and applications.

Servicing the Library OS

Once a month, on "Patch Tuesday," Microsoft issues security bulletins (MSRCs) describing software security vulnerabilities and releases patches to those vulnerabilities. The amount of code inside the library OS is smaller than the code in the full OS, so one would expect fewer patches to affect the library OS. This intuition is validated by an analysis of which binaries were affected by each security-related patch from 2006 to 2010.

As Figure 9 shows, the library OS requires significantly fewer security updates due to its smaller code size. Compared with Windows 7, Windows Server 2008 R2, and Windows Service 2008 R2 Server Core, the Drawbridge library OS has a much smaller servicing footprint. Of 286 security bulletins, 254 have resulted in patches to OS components. Just 36% of patches, 92, affect the library OS; the rest affect code found only in the host OS. Furthermore, some of the patches in the library OS might be redundant if they prevent one application from

compromising another. The comparison with Server Core installation of Windows Server 2008 R2 is interesting because Server Core is positioned as a way to reduce the attack surface of servers by removing "client" components such as the GUI shell. Drawbridge is affected by roughly half as many MSRC issues as Server Core over the same time period.

7. Related Work

This section surveys related work, primarily: previous library OSes, which adopt similar designs in service of different goals, and virtual machines, which provide many similar benefits at a higher cost. We briefly discuss other approaches to achieving strongly isolated processes.

Previous Library Operating Systems

Anderson [3] initially proposed that larger portions of the OS be factored into application libraries, primarily to improve performance. A general-purpose OS often applies resource management heuristics that can work at cross-purposes with a particular application. For instance, file read-ahead is a common optimization that is useful in the common case (sequential file access), but can harm performance of an application that accesses its data randomly. This idea was brought to fruition in the Cache kernel [10], Nemesis [21], and Exokernel [13]. The Cache kernel design focused on allowing applications fine-grained control over thread scheduling and memory swapping. The Nemesis OS was particularly focused on eliminating scheduling and performance interference through shared resources, in order for multimedia applications to meet real-time deadlines. The Exokernel design is closest to the bare hardware interface of a modern VMM.

Although previous library OS systems focused on performance, rather than encapsulation of the OS personality, Drawbridge shares with them several design points. At a high-level, the Drawbridge design is similar to the original Exokernel design, which had a kernel responsible for controlling access to the low-level hardware, and the rest of the functionality in the application library. The interface between the Exokernel and library OS was much closer to the hardware interface of a modern virtual machine, whereas Drawbridge provides higher-level APIs for threads and virtual memory, similar to the Cache or Nemesis kernels. The Exokernel also allowed applications to load constrained (or "safe") extensions into the kernel for management of hardware resources that could not be reasonably exported into a non-privileged address space. For instance, Exokernel library OSes would install packet filters into the lower-level kernel to multiplex the network. In contrast, both Drawbridge and the Cache kernel delegate low-level resource management to the kernel and neither system permits a library OS to load code into the kernel.

Relative to previous library OS designs, the first contribution of our work is to show that a large commercial OS can in fact be refactored into a library OS; this is in contrast to the minimal library OS implementations of previous systems. Second, Drawbridge incorporates an ABI design that streamlines the library OS implementation without compromising security isolation or unduly exposing implementation details of the underlying host OS. As a proof of this point, Drawbridge is the first library OS system of which we are aware that supports

either process serialization or migration of the guest application across diverse host systems. Finally, the Drawbridge ABI presents higher-level abstractions that trivially share host OS resources, such as the CPU and buffer caches. We believe the Drawbridge ABI offers a better starting point for library OS construction.

Virtual Machines

Virtual machines [35] are the primary success story for packaging an application and its dependencies on a fully-featured OS into a single self-contained unit. Research systems, such as The Collective [31], use a single virtual machine per application to encapsulate an application and its OS. Treating a complete legacy operating system, including the kernel, as a library for a single application wastes substantial memory and computation. This waste comes from an "impedance mismatch" that causes lost sharing opportunities, lost statistical multiplexing opportunities, and allocation strategies designed for physical resources running against virtual ones. Only through significant research and development effort has this waste been reduced.

New *paravirtualization* approaches, which warp the VM interface gradually farther away from a raw hardware interface, expose instead resources that a guest operating system can use more efficiently [5, 12, 17, 38, 39]. Despite significant research effort, however, running a full legacy operating system in a VM still incurs substantial overhead, as the legacy operating system brings with it system management processes and large allocations of kernel memory. Each guest can demand gigabytes of disk storage and hundreds of megabytes of physical RAM (perhaps reduced by clever sharing). Ultimately, paravirtualization is only chipping away at the margins; paravirtualized VMs will not scale to the same level as OS processes without more drastic measures.

A key contribution of Drawbridge is judicious selection of extremely paravirtualized VM abstractions in the Drawbridge ABI. This ABI demonstrably alleviates the overheads of hardware virtualization without unduly constraining OS design or compromising isolation. Roscoe *et al* [30] argued for research into higher-level hypervisor abstractions, Drawbridge answers that call.

Other Approaches to Library OS Goals

OS customization. Like previous library OSes, Libra [2] allows applications to customize their OS within a virtual machine in order to improve application performance. In Libra, an application that needs to extend the OS runs in a separate VM. The default OS functionality is provided by a sibling virtual machine over a shared memory protocol, and applications can selectively service their own requests from a custom implementation.

The performance concerns of Libra and other library OSes are largely complementary to Drawbridge; one could imagine customizing versions of the Drawbridge library OS with application-specific performance optimizations. More than optimizing performance, Drawbridge is concerned with encapsulating the personality of a library OS and enabling applications to run on future host OSes.

Xax and NaCl. Recent research on isolating untrusted web applications demonstrated that a limited set of OS abstractions were sufficient for porting large libraries of legacy code, albeit not entire interactive GUI applications [11]. The isolation technique proposed in Xax, namely hardware memory protection and a limited system call table, is shared by Drawbridge; NaCl [40] uses alternate techniques based on software fault isolation.

Stronger isolation in monolithic kernels. The monolithic organization of conventional operating systems makes them difficult to secure. In order to avoid the loss of functionality and performance, a number of systems have attempted to augment monolithic systems with additional security checks in a minimally disruptive manner. For instance, the Linux VServer [32] adds additional access control list checks and imposes a separate namespace for kernel objects used by an isolated process. All kernel data, however, reside within a shared, monolithic kernel address space. This approach of adding security checks and/or replicated kernel data structures is also exemplified by SELinux [23], zones [29], jails [34] and containers [7], however it suffers several drawbacks.

First, it is difficult to know that the job of inserting additional checks is done, or that the set of hooks is complete in a complex and rapidly growing kernel. Second, the abstractions to which the hooks must be added are exactly those that were not designed to express restrictions; therefore, the resulting mechanisms may not lend themselves to a reasonable policy calculus. For instance, a simple policy such as "Do not allow the contents of file X to leave the machine" must be translated into a perilous series of decisions on whether to allow innocuous-looking accesses to local system resources [15, 41].

It is tempting to try to paint a layer of isolation functionality onto an existing monolithic system, but the result is hard to trust since it requires maintaining properties across a large interface to a large and evolving code base. Often, the structure of a monolithic system makes it difficult even to rigorously specify isolation. Therefore, Drawbridge avoids a monolithic organization in favor of a library OS design, which lifts much OS functionality into the application's strongly isolated address space.

We hypothesize that library OSes will prove a more amenable platform for strong security isolation than a monolithic kernel, with a potentially simpler mapping of policies onto concrete decisions that must be made in the security monitor. Previous systems have explored policies much more thoroughly than this work, and we defer their design to future work.

Compatibility through API emulation. Several research systems have attempted to provide compatibility with legacy OSes by emulating their APIs [4, 16, 18]. Although emulation can work for common, heavily used functions, it is highly prone to subtle inconsistencies in the less used functions. Emulating the Windows API on Unix has required a colossal effort and consistently remained several releases behind Windows in terms of compatibility [30]. Even among Unix implementations, calls such as `setuid` are prone to compatibility issues [9]. In terms of compatibility, there is simply no substitute for bundling an application with the appropriate components of the original OS.

Application Virtualization. The need to robustly package applications without the high

cost of VMM solutions has driven development of at least two commercial products. Both App-V [26] and ThinApp [37] virtualize calls to the file system and registry. By doing so, they allow users to run desktop application like Microsoft Office without running a setup program. Because they virtualize only a subset of the file system and registry, they are easily circumvented or broken by new OS releases, and provide no migration for live applications. However, the application "sequencing" techniques used by these systems to create application packages are complementary to Drawbridge.

8. Discussion

The library OS allows a new model for packaging applications and new opportunities to reshape the computing experience. The fundamental problem of application packaging is deciding how to separate an application and its dependencies from their environment so the application can run predictably across a nearly infinite variety of user deployments. An ideal packaging technology makes it easy to deploy an application, continue the application across changes such as an OS upgrade, relocate the application from one computer to another, prevent the application from corrupting its host, and prevent the host from corrupting the application.

The library OS offers a new way to package applications. Applications can either be packaged with their dependent library OS, or can contain metadata informing the host OS of their required library OS. With the latter approach, a vendor could distribute and service a number of library OS images—say Windows XP, Windows Vista, and Windows 7 library OSes—with a single host OS.

For researchers and developers, the library OS approach offers the ability to rapidly evolve OS kernels, shells, and APIs independently without breaking other OS components or applications. Easily serialized applications offer opportunities to improve debugging and testing through techniques such as time-travel debugging, easy cloning of running application instances, and easy release of experimental OS components. For example, colleagues at Microsoft Research have used Drawbridge to deploy prototypes of new Win32 APIs that enable distributed user experiences.

Application Compatibility

Maintaining compatibility with application binaries across multiple OS releases is a significant challenge. While OS vendors expend significant resources maintaining compatibility, each OS release inevitably results in a large number of broken applications. Techniques for maintaining application compatibility in Windows 7 include one-off code in APIs; application compatibility shims, which load as intermediate code between application and API to alter input or output parameters; and Windows XP mode, which is a complete VM image of the latest release of Windows XP.

Application compatibility is particularly onerous for enterprises where one-off line-of-business (LOB) applications are abundant. The recourses available to an enterprise when a

LOB application is broken by a new OS release are often limited as programming staff has moved to other projects and source code may be missing.

By encapsulating the portions of the OS most likely to break application compatibility, the library OS offers a new technique to resolve application compatibility problems. Each application could run with the library OS of its choice. Programmers or IT administrators could bind an application to a known good library OS, which would continue to run correctly on top of new host OS releases. For example, an application packaged with a Windows XP library OS could continue to run correctly on Windows XP, Windows Vista, or Windows 7 host OSes. Conversely, an update to Windows 7 could ship with library OSes for Windows XP and Windows Vista. Users could upgrade to new OS releases unconstrained by the complex calculus of application compatibility.

Applications might even run on down-level OS releases. For example, an application with the Windows 7 library OS could run on Windows XP. Reverse compatibility is particularly useful for developers of new applications as they can target the library OS with the latest features rather than anticipating adoption curves.

Dynamic Application Relocation

With process serialization and deserialization, a running application no longer needs to be tied to the OS instance on which it started. An application could be moved to a new OS instance by serializing its state, moving the state to a new computer as needed, and then deserializing the state to restore the application. Drawbridge enables new opportunities for distributed computing, cycle harvesting, and applications running across OS-servicing reboots.

We are currently experimenting with a distributed computing environment based on Drawbridge in which running applications move from computer to computer to follow the user. For example, an application might start running a user's desktop, then move to the user's smartphone during their commute (which might be facilitated by dynamic instruction-set recompilation), and then move to the user's home computer. Running applications might also move from a laptop or smartphone to the cloud or a local server to conserve battery. While such migration is possible in theory with VMMs, VM images are too large to make it practical.

Process checkpointing has long been used to harvest spare CPU cycles in systems such as Condor [22] by moving background computations to under-utilized workstations. While systems like Condor were limited to applications without interactive user interfaces, with Drawbridge mainstream applications could be moved under the control of a resource manager system to create a "virtual desktop cloud" within an institution. Users could connect to their mobile applications through RDP proxies. Running applications could be cloned to multiple machines or staged to disk, to improve availability and to accommodate hardware failures or periods of peak load. With Drawbridge's per-application granularity, a virtual desktop infrastructure could be created much more cheaply than with VMM-based alternatives.

Serialization and deserialization of running processes offer end users new functionality even if applications are never migrated to a new machine. For example, instead of closing applications

for an OS reboot, the applications could be serialized to persistent storage and then restored after the reboot.

Reducing Security Risks

The modern Internet is a place of nearly constant danger. Technical and social attacks often exploit the fact that OS security is oriented toward protecting users from each other, and not from their own software. While the risks of running untrusted software can be mitigated through the use of "sandbox VMs", in practice the resource and administrative costs of a VM image are generally too high to afford each application its own sandbox, so typical users do not employ VMs to isolate software [19]. With the substantially lower overheads of the Drawbridge library OS design, every desktop application could be run within its own sandbox. Running a sandbox per application would mitigate many exploits that attack specific application weaknesses because the exploit could damage at most the single application in the sandbox. Drawbridge might also reduce a large class of social exploits—those which trick users into downloading programs—if downloaded programs always ran within a sandbox with a read-only copy of the library OS.

Cloud hosting services, such as EC2 and Windows Azure, might use the library OS design to substantially lower their per-sandbox costs. While VMMs offer the benefits of a complete OS, and thus will likely always have their place in server consolidation, the library OS uses far fewer resources and thus offers lower costs, particularly for cloud applications with low CPU utilization.

9. Conclusion

This paper provides the first existence proof that a feature-rich operating system can be refactored to produce a library OS that is compatible with desktop applications. Drawbridge provides a Windows 7 library OS that can run a large set of rich desktop and server applications such as Excel 2010, PowerPoint 2010, Internet Explorer 8, and IIS 7.5. With the exception of changes to licensing code, Windows 7 applications run in Drawbridge using unmodified binaries.

We described our heuristics for separating the "OS personality," as captured in the API implementation layer, from the rest of Windows 7 to produce a library OS that is 1/50th the size of the full desktop OS, yet still provides rich application compatibility. We believe our techniques are efficient, as we produced the entire Drawbridge system in less than two person-years, and that the same techniques could be applied to other operating systems.

We described our implementation and demonstrated through experiments that Drawbridge can provide increased system security, more aggressive system evolution, and greater application mobility through process serialization and deserialization. We have provided experimental data showing that the library OS design offers significant scalability advantages over VMMs. We demonstrated the improved flexibility for independent evolution by running the same library OS across four host OS releases.

The library OS is very relevant today. It offers new opportunities to significantly improve

end-user computing: redefining application packaging, increasing application compatibility across OS releases, enabling running applications to persist across OS reboots and relocations, and reducing the security risks of running untrusted software.

Acknowledgements

The original version of this paper was published in ASPLOS 2011 with Donald E. Porter, Silas Boyd-Wickizer, Jon Howell, and Reuben Olinsky as co-authors. This version has been edited for length and includes updates to some figures and text to account for changes in the Drawbridge system since the original publication.

References

[1] Amazon. *Amazon Elastic Compute Cloud (EC2)*. Seattle, WA, 2006.

[2] Ammons, G., Appavoo, J., Butrico, M., Da Silva, D., Grove, D., Kawachiya, K., Krieger, O., Rosenburg, B., Van Hensbergen, E. and Wisniewski, R.W. Libra: A Library OS for a JVM in a Virtualized Execution Environment. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, 2007.

[3] Anderson, T.E. The Case for Application-Specific Operating Systems. In *Proceedings of the 3rd Workshop on Workstation Operating Systems*, 1992.

[4] Appavoo, J., Auslander, M., Da Silva, D., Edelsohn, D., Krieger, O., Ostrowski, M., Rosenburg, B., Wisniewski, R.W. and Xenidis, J. Providing a Linux API on the Scalable K42 Kernel. In *Proceedings of the 2003 USENIX Annual Technical Conference*, 2003.

[5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[6] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A. The Multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, 2009.

[7] Bhattiprolu, S., Biederman, E.W., Hallyn, S. and Lezcano, D. Virtual servers and checkpoint/restart in mainstream Linux. *SIGOPS Operating Systems Review*, *42* (5), 2008.

[8] Bugnion, E., Devine, S., Govil, K. and Rosenblum, M. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. *ACM Transactions on Computer Systems*, *15* (4), 1997.

[9] Chen, H., Wagner, D. and Dean, D. Setuid Demystified. In *Proceedings of the 11th USENIX Security Symposium*, USENIX Association, 2002.

[10] Cheriton, D.R. and Duda, K.J. A Caching Model of Operating System Kernel Functionality. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation*, 1994.

[11] Douceur, J.R., Elson, J., Howell, J. and Lorch, J.R. Leveraging Legacy Code to Deploy Desktop Applications on the Web. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, 2008.

[12] Eiraku, H., Shinjo, Y., Pu, C., Koh, Y. and Kato, K. Fast Networking with Socket-Outsourcing in Hosted Virtual Machine Environments. In *Proceedings of the 24th ACM Symposium on Applied Computing*, 2009.

[13] Engler, D.R., Kaashoek, M.F. and O'Toole, J., Jr. Exokernel: an Operating System Architecture for Application-Level Resource Management. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[14] Franke, H., Russel, R. and Kirkwood, M. Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux. In *Proceedings of the Ottawa Linux Symposium*, 2002.

[15] Garfinkel, T. Traps and Pitfalls: Practical Problems in System Call Interposition based Security Tools. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2003.

[16] Gerard Malan, R.R., David Golub, and Robert Brown. DOS as a Mach 3.0 Application. In *Proceedings of the USENIX Mach Symposium*, 1991.

[17] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A.C., Varghese, G., Voelker, G.M. and Vahdat, A. Difference Engine: Harnessing Memory Redundancy in Virtual Machines. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, 2008.

[18] Helander, J., *Unix under Mach: The Lites Server*. Helsinki University of Technology, Helsinki, 1994.

[19] Howell, J., Hunt, G.C., Molnar, D. and Porter, D.E., *Living Dangerously: A Survey of Software Download Practices*. MSR-TR-2010-51, Microsoft Research, 2010.

[20] Keetch, T., *Escaping from Protected Mode Internet Explorer – Evaluating a potential security boundary*. Verizon Business, London, UK, 2010.

[21] Leslie, I., McAuley, D., Black, R., Roscoe, T., Barham, P., Evers, D., Fairbairns, R. and Hyden, E. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas In Communications*, *14* (7), 1996.

[22] Litzkow, M., Tannenbaum, T., Basney, J. and Livny, M., *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*. University of Wisconsin-Madison, 1997.

[23] Loscocco, P. and Smalley, S. Integrating flexible support for security policies into the Linux operating system. In *Proceedings of the 2001 USENIX Annual Technical Conference*, 2001.

[24] Love, R. Get on the D-BUS. *Linux Journal*, 2005.

[25] Microsoft. *Internet Information Services 7.5*. Redmond, WA, 2009.

[26] Microsoft. *Microsoft Application Virtualization (App-V)*. Redmond, WA, 2006.

[27] Microsoft *Performance Tuning Guidelines for Windows Server 2008 R2*, Redmond, WA, 2009.

[28] Microsoft, *Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification*. Redmond, WA, 2010.

[29] Price, D. and Tucker, A. Solaris zones: operating system support for server consolidation. In *Proceedings of the Large Installation Systems Administration Conference*, 2004.

[30] Roscoe, T., Elphinstone, K. and Heiser, G. Hype and virtue. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems*, 2007.

[31] Sapuntzakis, C., Brumley, D., Chandra, R., Zeldovich, N., Chow, J., Lam, M.S. and Rosenblum, M. Virtual Appliances for Deploying and Maintaining Software. In *Proceedings of the Large Installation Systems Administration Conference*, 2003.

[32] Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A. and Peterson, L. Container-based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ACM, 2007.

[33] Spear, M.F., Roeder, T., Hodson, O., Hunt, G.C. and Levi, S., Solving the Starting Problem: Device Drivers as Self-Describing Artifacts. In *Proceedings of the EuroSys 2006 Conference*, Leuven, Belgium, 2006.

[34] Stokely, M. and Lee, C. *The FreeBSD Handbook 3rd Edition, Vol. 1: User's Guide*. FreeBSD Mall, Inc., Brentwood, CA, 2003.

[35] Sugerman, J., Venkitachalam, G. and Lim, B.-H. Virtualizing I/O Devices on VMware Workstations Hosted Virtual Machine Monitor. In *Proceedings of the 2001 USENIX Annual Technical Conference*, 2001.

[36] Torre, C. Mark Russinovich: Inside Windows 7. *Channel 9*, Redmond, WA, January, 2009.

[37] VMWare. *ThinApp*. Palo Alto, CA, 2008.

[38] Waldspurger, C.A. Memory Resource Management in VMware ESX Server. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[39] Whitaker, A., Shaw, M. and Gribble, S.D. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[40] Yee, B., Sehr, D., Dardyk, G., Chen, J.B., Muth, R., Orm, T., Okasaka, S., Narula, N., Fullagar, N. and Inc, G. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[41] Zeldovich, N., Boyd-Wickizer, S., Kohler, E. and Mazières, D. Making information flow explicit in HiStar. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, 2006.

A Letter

Joe Barrera

In undergrad, I was a math major, and really interested in programming language semantics, and so forth. But my good friend Brian Zill (now at MSR and also a CMU alum) was messing around with Unix—we had 4.2 BSD Unix on a few VAX 11/750s—and so I got really hooked into the whole kernel hacking thing. It just seemed like the Marine Corps or Particle Physics of Computer Science... The Elite. So when I was applying for graduate school, I heard rumors that BSD 4.4 (the next upcoming version of Unix used by universities) was actually not going to be done at Berkeley (the B in BSD), but rather at CMU... that CMU was the new center of the operating system world. Actually, I think I heard these rumors from Rick himself, on the phone, after I was accepted by CMU. I think this is the same phone call where he explained that his name is pronounced RA-shid and not ra-SHEED.

And sure enough, I saw papers about Mach as a new foundation for Unix. I visited the MIT and CMU campuses, trying to figure out where to go to grad school... but at that point I was hooked on the New Unix vision, so CMU it was. And I sort of envisioned that whomever was leading this New Unix mission at CMU would be a hard-core, hard-ass, marine-style sort of leader... but then I met Rick and he was this laid-back, happy, optimistic guy... nothing ever bothered him... Rick was a puzzle! (Later on, of course, I would meet Dave Cutler.)

So I have learned a LOT of things from Rick that have been *immensely* important to me—things that, like for many people here, have shaped my career and my entire outlook on life.

One of the first lessons I learned came from the very first Mach project I worked on—Distributed Shared Memory. Now, this concept had been implemented by others, following from Kai Li's thesis work at Princeton.

But Rick said, Mach should be able to do this easily, so why don't you whip up some code

and make it happen. So, I said OK, and to myself I thought: what are the absolute hardest parts of this problem? How can I make this as difficult as possible? Well, let's see... I guess page sizes could be different on different computers... I guess I could try to totally optimize the read/write scheduling of pages... so I'm going to sit down for a month and try to find some optimal algorithm on paper that addresses these issues. I was trying to make this as complicated as possible. (And I accidentally wrote some code that worked in the meantime.) And then I remember very clearly sitting in my office, and Sandro (Alessandro Forin) was there! And he said, "well, I did these measurements, and I made these changes to your code..." and I'm like (in my mind)... Wait a moment! This is MY code, this is MY project, OK?! But I learned at this point that you just need to DO it—you can't just sit still and do the whole "analysis by paralysis" thing—you have to do the project, you need to address the *important* points, you have to get it OUT there, because the rest of the world won't stand still. And the things you are obsessing about are probably not important at all.

That was a real wake-up call for me. And I owe both Rick and Sandro for teaching me that lesson.

The second lesson—well, not really a lesson—was how to take a huge, impossible-to-understand mass of code, and understand enough of it, incrementally, to make it work. In this case, it was integrating the NFS and AFS file systems into Mach. The previous student who was working on this kind of ran away screaming from this project... I think he changed advisors, or fields, or both, after that... OK, so I take this stuff, and it sort of compiles and it sort of works a little bit, but I'm like, how can I possibly do this? There's no one to explain what it's supposed to do, there's no documentation, there are fourteen different versions of the code... so I start working on it and I find myself going into Rick's office and asking him, "where's the part that does *this*"? And he'd smile, and he'd open up Emacs, and he'd search and grep and edit and grep, and he'd say... oh! it's here. After doing this a few times, I realized that everything I was asking Rick... he didn't know either! And he was just using methods to find the answers that I could have used myself—and so I learned those methods and left him alone. In retrospect, maybe he did know the answers, but he did me the favor of showing me how *I* could find those answers. And this skill, of taking a gigantic blob of undocumented code and figuring it out and mastering it, I've applied over and over and over in my career. Right now at SLAC I'm dealing with this huge glob of undocumented C++... written by physicists... with no, I mean literally NO comments... (I spent the first two months getting all the segfaults out)... but you have to have the confidence to mechanically go through the code and figure out that this does that, that does this, because... as Rick always said... it's just code.

The final lesson I learned from Rick is: don't be afraid to change how things are done—and if you want to change how things are done, demonstrate what's possible with actual implementations and existing semantics. Certainly that was one of the big lessons from Mach—that Unix can be implemented in wildly different ways. Some of those ways have been adopted by modern Unix (Linux or OS X) and some haven't, but Mach expanded the space of what was possible and practical in the implementation of operating systems—while preserving the semantics and performance that users were used to. (Other OS projects at the

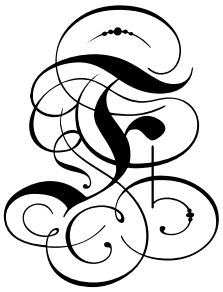
time—Sprite, V, Plan 9—explored new ideas but at the cost of incompatibility). On a smaller scale, I've applied this lesson to projects I've been responsible for. Most recently, when I was at IBM, I fixed a major pain point in a Tivoli product by moving from a multiple database solution to a single database solution. Multiple separate databases were used because the product was geographically distributed.

But when it came time to aggregate results into a single reporting database, the product would be unavailable while the copy was occurring.

As implementations scaled up, the copy time would approach 100% of uptime. Shifting to a single database was a hard sell—everyone knew it would be slower and that the single database would be a bottleneck.

Except—that it wasn't. And the only way of demonstrating that was to write a prototype, making changes throughout the code base (see lesson 2) and demonstrating the prototype working in a completely compatible form with the existing implementation (lesson 3). We actually shipped this new architecture—in a point release—TADDM 7.2.1 Streaming Server. Customers have since asked we didn't call it 8.0... but that's a marketing question. (Something I *didn't* learn from Rick.)

—Joe



ounding Microsoft Research

Letter to Rick Rashid

Bill Gates

When I first got e-mail about the Festschrift I wasn't sure what it was. Then someone mentioned that it involved Rick being 60 and that really surprised me.

No one I know is as open minded to new or ideas or more willing to try things out by writing code himself, so it is hard to believe he is 60.

However, I am very happy to add my voice to the many others who are congratulating Rick Rashid on his 60th birthday. I don't know that Rick puts much stock in milestone birthdays like this, but it provides a good excuse to step back and acknowledge his achievements.

Of the many achievements of Microsoft over the years, the establishment and growth of Microsoft Research stands out for me as a singularly important accomplishment, and Rick has been key to its growth, essentially, every step of the way.

When we established MSR in 1991, we were one of the first software companies to launch its own dedicated research organization. In those early days, as Nathan Myhrvold and I looked out on the horizon of research all-stars that we wanted to recruit to the company, Rick stood out for his remarkable breadth of scope and standing in the field of computer science. We had to ask him three times, but we were finally able to convince him to leave Carnegie Mellon University and come to Redmond.

Under Rick's leadership, MSR has grown from a small group of researchers in the Northwest corner of the U.S. to a truly international organizations, with 850 researchers involved in over 60 areas of computer research around the world. Today our labs in Cambridge (Mass.), the Silicon Valley, Cambridge (England), Beijing and Bangalore stand witness to Rick's vision, leadership and prodigious recruiting talent. The organization he has built and led has few—if any—equals in the world of technology.

More than just locations, Rick has created and nurtured an environment at MSR that encourages

collaboration and cross-disciplinary investigation. Rick has even had a hand in our building design, to make certain that MSR provides the best possible work environment to advance the state of the art in computer technology. (And by the look of Rick's own office, apparently the presence of a large number of pieces of Star Trek memorabilia is particularly conducive to advanced research.) The contribution of this group has touched every product Microsoft ships today, and is leading the way for new approaches to some of the most important technology questions we face. My confidence that Microsoft will continue to develop breakthrough software comes in large part from the strength of the MSR organization Rick has built.

Many companies that built strong research organizations like AT&T, and Xerox didn't manage to connect them to their product groups. Rick made sure that was not the case at Microsoft.

Not only has Rick made sure we bring in great people but he has personally contributed hugely to Microsoft innovation. Whenever I am unclear on where technology is going, Rick is one of the people whose thoughts I trust.

Rick's great passion for technology, his leadership and vision have all been instrumental to the creation of this important institution. Technology users around the world – whether they know it or not – owe so many breakthroughs to the impressive work of Microsoft Research, all of it enabled by the contributions of Rick as its leader and mentor. I look forward to that passion and leadership animating MSR for many more years to come.

Bill Gates

March 6, 2012v

Anecdotes from MSR's Founding

Nathan Myhrvold

9 March 2012

In 1990 I started scheming as to how Microsoft could have a research lab. At the time that was kind of a crazy proposition. No Silicon Valley company had a research lab, even though all of Silicon Valley sort of came from Xerox PARC, which was a research lab. And the contradiction was explained in a book called, "Fumbling the Future," which explained to everyone that Xerox shouldn't have done this research lab even though that's what gave rise to personal computers and everything else.

So it was kind of a strange environment to be asking for a research lab in, but I persevered, and I wrote a big memo on it. I convinced Bill and then everybody else, and so we were in business to have a research lab, only we needed somebody to run it.

So, I met Gordon Bell, and Gordon was both a guy who had a great history in research and he just might be crazy enough to help out on this.

The second part was certainly true, he was crazy enough and he helped. Gordon and I went on a pilgrimage around the country meeting with some of the great names of computer science to see if we could convince them to come to Microsoft and run research.

It was interesting, because a lot of the people were very famous, I had read their papers, I knew all about them. And we would come in, and Gordon and I would walk out, shaking our heads like, "Oh my God, no way, no how."

Sometimes they would tell us up front that they weren't interested. Of course, that was presupposing we were interested, we were interested in talking to them.

Finally, we decided to go see Rick Rashid at CMU. And he was maybe halfway through the whole process and, Pittsburgh isn't really on the way to anywhere except Pittsburgh, so we went to Pittsburgh and we met with Rick.

And Rick was fantastic. He was funny, he had a great research track record. He had a property that I would actually find he kept for years and years at Microsoft also, where he could talk about very high-level computer science things, but he would also tell you how many instructions a loop took, because he knew, because he actually wrote the code. Now, I actually wrote code so I thought that was kind of cool.

A lot of these other people had gone on to what they viewed as a higher plane of existence where they could be a head of a department or a dean or even in a big research lab, and not have to do petty little things like write code.

That always annoyed me because we wrote code for a living at Microsoft, and if you don't love writing the code, how are you going to actually do anything else?

Rick clearly loved writing code, but he also still could be an academic of the first order as opposed to many of these guys who thought that being a great academic and a great researcher or a great research administrator meant that they shouldn't ever get their hands dirty.

So, Gordon and I walked out of the meeting with Rick, and we're like, "Yes, this is the guy."

Now, Rick was polite to us and was a little bit amused with the whole thing. Of course, if you know Rick, you know that that's often the way Rick seems, a little bemused with the whole thing, because I think Rick is bemused with pretty much everything.

But he set our expectations pretty low. Nevertheless, Gordon and I were high-fiving it all the way out to the car. We drove like lunatics to get to the Pittsburgh airport so we wouldn't have to spend the night there. And the whole way back we think, "This is great, we found the guy. We're going to talk to a couple other guys but we found the guy."

So, we made a formal offer to Rick, and he turned us down.

I called him up and I was like, "Surely I can talk him out of this." He turned me down again.

I was at a bookstore—actually back in the early '90s we had these things called bookstores. I remember I bought this book; I still have it. It's called "*Getting Past No: How to Deal with Difficult People*." Well, I went through that damn book and mostly it was about a kind of guy that was difficult, but wasn't the kind of guy Rick was, so it wasn't really there.

And so Gordon and I got together and I started talking, he started talking. Of course, Gordon interrupted me, then Gordon interrupted himself, and we got in a fever pitch at one point and Gordon said, "We can't let him make this mistake; we owe it to him."

So, we said, "Okay, let's figure out what to do." So, we each wrote a letter to him and Gordon was particularly heavy-handed in his letter because he felt he could be, pointing out that if you work successfully in technology you can do all kinds of crazy things like start computer history museums. Whether that was the line that actually made the difference or not I'm not sure, but it was in there.

Then we got Rick to reconsider, and that is one of the single best hires I've ever made in my life for hiring somebody. It was the single best thing I ever did for Microsoft Research.

As soon as Rick joined Microsoft Research, we all of a sudden had to get other people to join Microsoft Research. I'd actually hired a speech recognition group into Microsoft Research before Microsoft Research existed, so we had a few folks already. I had a group called Advanced Development, and so we had a few people there, but we really needed to build a research team.

So we then embarked on a very similar sort of pilgrimage to a bunch of places that Gordon and I had done to say, "Well, who else can we recruit and how can we get them to come here?" And some of those people said no, much to their later regret, but gradually we found people that we could convince to come.

One of the principles that Rick and I developed early on in Microsoft Research is that we wanted to hire people who I used to say were "narrowly insane," meaning that they wanted to solve a really hard ambitious problem, a problem that other people might think is kind of crazy, you couldn't solve it. You wouldn't want them to be so risk-seeking that they were totally wacko, you just want them to think that their problem was something they could make progress on.

And one of the interesting things in managing research as a result, something Rick became a master of, is, when one guy's working on his problem and it's really hard but he thinks he can solve it; and this other person thinks she can solve her problem. Well, if you ask the one person about the other one's problem, they say, "Oh shit, no, that's hard, are you kidding, oh my God, do you know how many people have failed at that?" The same person who knows in their heart that they can make their problem work is deeply skeptical about everybody else's. That's what I meant by narrowly insane. They have to be sensible enough that they can make progress and be generally sensible people, but they have to be just crazy enough to think they can do something that no one Earth has ever done before. Microsoft Research is a huge collection of those, organized in a way that we could really make some progress.

My favorite hire in that whole era was actually Jim Kajiya. Before Jim we would have people come and mostly they were people I knew, or people Rick knew. We would heavily twist their arms and we'd pitch for all we were worth and talk about the wonderful future and how it was going to be so fantastic. It was this very intensive thing, very future-tense oriented. We couldn't sell them on the present, we had to say, "Hey, we're going to build this great thing, just *imagine* it."

So, Jim Kajiya was on our list, but what Jim wanted to do was come and hang out for a week at Microsoft Research. I said, "Shit, Rick, can we do that? I mean, do we have enough of a Microsoft Research? Isn't he going to see that the Emperor has no clothes? We only have half a dozen, a dozen people, some of them are still unpacking their boxes; this is a disaster." Rick said, "No, I think it will work." I said, "Shit, if it is a disaster, he's going to tell all these other people—well, I won't be able to go to SIGGRAPH again, because people will all snicker when I come in the room," which frankly they were doing before that. "So, is this really going to work?" Rick had some reservations but he said, "Well, we'll try it."

So, Jim was there for the week, and he was scheduled to see me at like 3:30 on Friday. And

I was hearing from Rick and from Dan, who was also trying to manage the whole thing, "We think it's going okay." It was like, "Well, you know, we seem to like this guy but we're not sure."

So, Jim comes in to see me, and I said, "Well, how did it go?" He said, "Well, I thought I'd just cut to the chase and try to convince you of the five reasons you ought to hire me." I thought, "God damn, that is so excellent." I made it seem like he was trying to sell me a little bit, but, in fact, we had gotten over the hump enough that a guy like Jim Kajiya, who was at the top of his game, who had a great position at Cal Tech, after spending a week with us, actually wanted to be there. We were moving from being an organization that was all about the future tense into something where we had at least a little bit of a present tense, so, that was a terrific moment.

As we started to build Microsoft Research out to the point where we got research publications, the first year that we really had a big showing at SIGGRAPH, we had gone from like nothing to about 25 percent of the conference. Then as other key conferences came up suddenly we had people who were there, and there was this amazing sense that it was going to work. So, as Microsoft Research progressed and grew, we got more adventuresome with the things that we would research. Initially we were focused on things that were really part of the core software mission that we thought Microsoft was about, but over time we said, you know, we can afford to spend some time on things that are a little bit more theoretical, a little bit more "out there."

In the early '80s I was a graduate student at Princeton in theoretical physics, and one thing about theoretical physics in general, but particularly *then*, is that there weren't a lot of girls in the class. But there were two in the class while I was there, and one was Jennifer Chayes. And although Jennifer certainly attracted the notice of everyone in the department, it was only notice from afar because Jennifer's husband was also in the class, and he was a punk. In every sense of the word he was a punk. Actually Jennifer was, too. She would come to parties wearing a black dog collar with big, thick, sharp metal spikes on it. So, you were very intrigued with Jennifer but you were also a little afraid, just flat out.

So, as we started looking at doing more mathematical and theoretically oriented computer science, I was, at the time, on the board of the Institute for Advanced Study, and Jennifer was visiting there. So, Jennifer and I talked, we talked about her research, we talked about a variety of other things. I went to visit her at UCLA where she also was, then again at the Institute, and I said, "Hey, Jennifer, maybe you should come to Microsoft." And, of course, to ask someone who (at the time) thought of herself as more of a mathematician was as crazy as it was to hire the original researchers into Microsoft in the first place.

But I persevered and I got her to come out there, and she got intrigued. And she said, "It's just that there's a problem: my husband. She said, "I've got this husband but he lives in Germany." I said, "Really, how does that work?" She said, "Well, not very well."

And, of course, while she's describing Christian to me initially, I'm thinking, yeah, she sure can pick 'em, let's see what this one is like.

But, in fact, we managed to hire both Jennifer and Christian to Microsoft, and they then went and hired a whole bunch of other people, and we managed to make Microsoft Research

an institution that was not just great at pounding at code, not that there's anything wrong with that, and not just great about things that are fairly *applied* aspects of computer science, making a better operating system or a better file system. It's pretty easy to see how that directly would benefit a customer. So, even if it's research, it's research that has a pretty short path to application. Not so with all aspects of abstract computer science and discrete mathematics, but, in fact, many of those ideas, even though they're more remote, are also more powerful, so they've become part of the whole Microsoft Research portfolio.

Because Rick really is this great combination of somebody who understands computer science at any level of abstraction that you can imagine, and who also really likes computers and really likes to code, he also, strangely, (and this is one thing that is quite different, I like computers, I like to code, too) likes people better than I like people. And so we'd have some frustrating personnel situation or another and Rick would always have a way of dealing with it, and he'd have this great set of stories. We had a big issue once with these people that were bickering, and so he told us this story of the imaginary nickel. His kids were fighting in the back of the car, and he said, "You know, you kids are squabbling so much, if I told I gave one of you an imaginary nickel, you'd fight over it," whereupon this is exactly what they did.

Well, that story was really useful in getting these two bickering, pretty high-level people at Microsoft to maybe consider they should stop bickering, and there was something more at stake than an imaginary nickel.

Another one of the favorite stories that Rick would pull out when we were talking to people about the pace of technology is the story of one of this sons that would ask him to tell a story before bedtime just pretty much as a way of delaying bedtime, as kids are wont to do. And then he'd ask a question. So, he'd say, "Yeah, well, here's a question or here's a question."

So, one night he's tucking him in and he said, "Dad, when you are my age, did they only have eight-bit computers?" And, of course, Rick has to explain that actually he was in college before he ever saw an eight-bit computer—you know, there's a hell of a thing, but to a child growing up then, you know, 32 bits was normal. So, no matter how antediluvian, no matter how ancient Dad was, he couldn't have been before the eight-bit era, right? But, of course, he was.

So, over the years, Rick has been an incredible friend, he's been an incredible researcher, he's built Microsoft Research into one of the great institutions in the world. And besides all the value that's given to Microsoft, Microsoft customers, Microsoft shareholders, that's very substantial; research has made an incredible contribution because the ideas in research have changed the lives of millions of people.

That was one of the lines, actually, I used on Rick to try to recruit him. It didn't work, but the line went like this: If you want to affect the lives of millions of people, Microsoft is the best place to do it, because if we have great ideas there we can get them more clearly, more directly into products. Well, it didn't work but then Rick came and he made the whole thing true, so he actually did affect the lives of millions and millions of people.

So, it's been an incredible experiment. This memo I wrote in 1991 sort of laid out all of the

things that research was going to do, and almost every one of them we did in the highest level sense. It says we'll open up one in Asia within 10 years; we did that. It says we'll open up one in Europe; we did that. It said we'd consider other parts of the world; of course, we did—I say "we"—after I left, India and a variety of other places, even places like *California* were added to show how far we would go in outreach.

I just can't imagine a better research executive than Rick Rashid.

Microsoft Research Plan*

Advanced Technology and Business Development

**This memo is the original document Nathan Myhrvold wrote in 1991 recommending the creation of a Microsoft Research and mapping out its structure. Historically, it was Nathan's brain child and the plan and proposal that he used to set up the development of Microsoft Research.*

1. Introduction

Once upon a time, it was very clear what our product agenda was—simply take the ideas which were successful on "big computers" and move them to "little computers". A great deal of effort and cleverness went toward engineering this feat—much of it directed at problems like living with too little memory, using awkward processors, and coping with the complexity of assembly language programming. Later on, we got a bit more sophisticated and the PC industry started developing some of its own unique innovations such as a graphical user interface. At present we have just about exhausted the store house of existing technology, and the days of taking something off the shelf and adapting it to our "little computers" are over. One reason is that we have already done most of it, and of course another is that microprocessor based systems aren't necessarily "little" anymore—they rank among the most powerful general purpose computers on earth.

The onward march of hardware technology is taking personal computers to new heights of processor speed, memory and general functionality per user. CPU cycles and power, however dramatic, are only one of the issues. We are also faced with the staggering potential offered by entirely new developments in other parts of the system, such as optical storage systems, new graphics capabilities, digital data broadcasting, digital video and a host of other hardware and software innovations which will change the landscape of our industry.

The challenge, of course, is to create the software which realizes the potential that this hardware offers. We have to explore and research the new software technology necessary for the PC environment of the future, evaluate the impact on our upcoming products, and then follow up with the leadership and investment of resources necessary to turn ideas into commercial reality. Microsoft has a great deal of experience with the last two stages, but historically we have not had to spend as much effort on the first. This is going to become increasingly important to us, particularly as we seek to broaden the realm of personal computing, and make it an even more important facet of people's day to day lives.

In very general terms, we have to invest in our future by doing more work in research and technology creation. The remainder of this report is on what we specifically should do.

2. What is Research?

The are a variety of different forward looking, high technology activities which Microsoft could and should be involved in. Here is a taxonomy of the different primary functions:

> Tracking the state of the art. We certainly need to keep abreast of the trends in our industry, and this requires having sufficient resources to monitor up and coming areas of interest, and determining their impact on our industry and our business. Recent examples of this work include analyzing the JPEG standard, reports of technology in Japan etc.
>
> Advising product groups. Another important function is advising product groups on technology. Often this means evaluating different technology options when a sudden need arises and the product group needs more expertise to evaluate the

situation. This has been a large part of what ATBD has done in the past, in areas such as device independent color and fonts.

Technology acquisition. It would be foolish to believe that Microsoft could develop all of the technology which we require, or that we would even want to try. It is important to be able to evaluate and acquire technology. When this technology is directed at a specific product, then it is clear that they should have responsibility for evaluation and making the deal. Often, it makes sense to buy technology outside a specific product group. In some cases, the technology is cheaper and easier to acquire if you do it early, before a product group feels the need (which was the case when we initially contacted SGI), or when the product is fundamental enough that it crosses many products (as is the case with some deals in progress).

Advanced development. This category includes projects which have a greater technology content, and risk, than normal product development. Sometimes the degree of risk and new technology is quite high, and the goal is to only to create a prototype, but in most cases advanced development seeks to produce working code (usually for productization). Advanced development is often best done in the same organization that does straight development projects.

New technology & business projects. There is a class of advanced development projects which includes both technological innovation in a new area for the company as well as interaction with external companies as either partners or customers. We currently have ongoing projects in this category. These are often more like a product in structure. An example project which we may do in the future would be working with a consumer electronics company on an operating system for intelligent HDTVs. A distinguishing feature of these projects is that they are oriented toward establishing Microsoft in new strategic arena rather than just being products in the normal sense.

Research. Finally, there is the task of working on unsolved problems in computer science which are critical to our strategic needs in the future. This is really applied research, because we would expect to incorporate this work into products within a 2 to 5 year time horizon.

ATBD has mainly focussed on the first five activities. Our plan for some time has been to expand the group, both to be able to handle more work in the existing areas and also to expand into new areas such as research. After looking into this in more detail, we have come to some different conclusions than in the past, and the need for a research group has become a much higher priority.

At one point we had the model that we would hire a set of "experts" in various areas—graphics, natural language, and others. The total number of such people would be fairly small—say 5—10—and they would help to keep us informed on new developments, advise us in their areas of expertise etc. This really covers the first two categories above, rather than actually doing research because you need to have greater resources and a specific agenda to

do research—i.e. each expert would need a group. Our experience is that is very difficult to attract top ranking technical people for these positions. The job is a combination of being an analyst, a consultant and a reporter. The best technical people are interesting in doing something active, rather than being an "armchair" researcher. The closest analog in Microsoft's parlance would be a program manager. People with the same qualifications as an MS program manager, given appropriate access to technical experts, would be great at this, but you can't get the experts to hang around just for this. Do-ers want something to do.

Another problem with this model is that it does not give us any ability to do research, and there is an intrinsic need for certain hard problems to be solved. In the course of the next couple of years our development organization will need access (and in some cases exclusive access) to technology which does not exist today. Although we are quite capable in creating products, our developers are not equipped for this task by themselves. This isn't just for extra credit, in many cases it is needed just to fulfill the vision and commitments that we have already embarked upon. There is a lot of technology that we can and should count on getting from outside—either university research or outside companies, but there are also some areas where we need to, or have an opportunity to, create new technology ourselves.

This has lead us to put more emphasis on building a group to focus on these problems. This would include world class experts, as well as sufficient other resources so that they could actively work on applied research problems which we select as high value contributions to our strategy. They would also be available as a resource to the company for the other categories above, but this would be a sideline rather than the primary job.

2.1 Advanced Development

One of the key distinctions which has to be made is the difference between doing advanced development, product development and research. Many companies confuse these issues to their detriment. We believe that Microsoft needs to have a full spectrum of development.

The table below covers several different points on that spectrum. The three key characteristics which can be used to distinguish them are the degree of technical risk and the relation to other projects or groups. Technical risk in this context does not mean the engineering task of writing the code and fixing the bugs, but rather the question of whether we know up front how feasible the project is, and whether new techniques will have to be developed in order to achieve our goals. This sort of risk is deliberately low for most of our product development—we do not usually set out to do a product without having some confidence that it can be done. There are always surprises, but by and large the Excel 4.0 team does not doubt that they can make a great spreadsheet, or that they will have to push the frontier of knowledge in computer science forward in order to do so.

At the other extreme, research is by definition aimed at solving problems which we do *not* know how to solve. Obviously we would not start them if we didn't have some confidence that we could make progress of some sort, but you must acknowledge that there is a fair amount of uncertainty. Sometimes you don't whether you will succeed, other times you feel sure that you can do something, but you are not quite sure what it will turn out to be. The biggest uncertainly is usually time—you might feel that the problem is soluble, but it is often

difficult to estimate *when* you will manage to solve it.

Activities	Technical Risk	Related Work	Group Involved
Current product development	Low. Uses known ideas.	Other products in business unit.	The "status quo" product development groups .
Future versions & concepts	Medium. Tries to apply new methods.	Follow on to existing product.	Product groups or advanced development groups in same business unit
New/ advanced product exploration & prototypes	Medium. Key ingredients exist, but need integration.	May be related to existing products, or not.	Existing product groups, new product groups dedicated to the area, advanced development groups.
Advanced technology used across products.	Medium—High.	More related to the technical topic than any one product.	Advanced development groups. Must be responsive to all "customer" groups.
New technology and business development	Medium—High. Usually there is no existing model to follow.	Primarily driven by technology in its domain, and by external partners.	Advanced development groups combined with business and technical strategy.
Research	High to Very High. New approaches must be invented.	State of the art research, where ever it is found.	Research group.

In between the two extremes are a number of forms of advanced development. Those which are closely related to product groups should certainly happen in these groups. The NT project is a classic example of something that started out as an advanced development project and which matured into a product effort. Other projects, like Pen Windows may be associated

with groups which are not directly building the base product. In this case, it is being driven by the Applications Division which wants to use Pen Windows for pen specific products rather than the Windows group itself. Multimedia Systems is a similar in this respect.

I believe that we need to increase the amount of advanced development work that happens in the company overall. This has direct benefits in terms of making our products more innovative and competitive. It is also a primary way that research work will find its way into products. Some work of this nature will be associated with the New Technology Projects group in ATBD, but in general most of it has to occur throughout the company. We need to think about how to stimulate this on a company wide basis. It is a very important issue, but beyond the scope of this report.

3. What's in it for the Company?

There are many examples of companies not getting very much out of their research departments. A lot of people ask , "What about Xerox PARC, or IBM's Watson Research Center, or Apple's Advanced Technology group?" There are many examples of good research which does not benefit the company involved, often to extreme proportions. Xerox is an example of a company that did the right stuff at the right time with the right vision, and still lost. They invented GUI, and yet it never did Xerox any good. IBM Research has won several Nobel Prizes for fundamental discoveries, but it is not at all obvious that IBM gets direct commercial benefit from research overall. The work they do is quite good— they invented high performance computer architecture (pipelining and practically every major idea except those due to Seymour Cray), optimizing compilers, RISC architecture, SQL and relational databases, the DES encryption scheme, arithmetic coding for data compression and other major advances. I think that the case can be made that a lot of benefit is derived from this work, and IBM does get *some* of it, but it is also quite clear that most of the benefit winds up going to others (Oracle in the SQL case, MIPS & Sun in the RISC case, the world at large for DES...). In the case of IBM and Xerox the work is good, but the connection to products is weak. There are also companies like Apple who spend a lot of money, but it is not clear to an external observer what, if anything, is going on. They have over 300 people in their advanced technology group, and have some amazing toys (their own Cray XMP for example), but nothing major has come out yet. Meanwhile, the advanced development work at Apple which has had a commercial impact, such as Hypercard, was done largely by one guy with a couple assistants—and he left the company last year. Maybe they are doing good work, but can't transition it to products, maybe they just play around—it isn't clear.

The first answer to this is that when it comes to accounting for success and failure, *research is no different than any other corporate activity*—there will always be some spectacular failures. Every aspect of business is mismanaged by somebody, and it is not at all surprising that research is among them. When people focus on the question of "why doesn't corporate research work?", and use examples like those mentioned above, they are almost always overlooking the fact that you could equally "prove" that finance, marketing, advertising etc don't work either. Look at start ups—you could point at the wreckage of a thousand valiant efforts and dismiss them too. As a class they are very risky, yet many do succeed—enough so that the PC industry is lead by companies which were start ups only 10 years ago.

The most famous examples of people not being able to transfer research to the development organization are actually not surprising when you look at it in detail. *In many cases there was no development group to speak of to give the work to!* This was certainly the case at Xerox PARC—the computer product side of Xerox was nowhere near as strong as PARC. It was not a case of having two top notch teams that couldn't agree—they basically could not get out of their own way as far as developing products was concerned. The same is true for Bell Labs—their best work was done at a point when it was *illegal* for AT&T to be in the computer business. More recently, this has changed, but AT&T is still not very competent in any sort of development or marketing of computers. If you can't do products at all, it doesn't much matter whether the inspiration was from research or not.

One of the very best examples of this has occured recently with Xerox. They bought Ventura, which was selling the leading PC desktop publishing package. Within a fairly short period of time, the original founders have left, and the moves that Xerox is making are clearly ruining Ventura Publisher and any hope that it had of continuing to be a major player in the PC publishing. If you take a leading application with an established market and a very solid product and then proceed to run it into the ground, how can you hope to productize and nuture a research project?

Another factor which is quite telling is that most computer companies which have been able to afford to fund research in the past were *hardware* companies. If you look closely at the record, they have usually done very well at integrating hardware advances into their product line. The IBM RS/6000 uses a multiply chip with a very innovative algorithm which was developed at Watson Research center. It is the primary factor in the excellent SPECmark performance they quote, which is heavily floating point intensive. Nobody else has anything like it—it is a clear case of taking research to the market in a timely and effective way and getting a decisive advantage because of it. The same thing is true for every major hardware company including HP and DEC. Xerox has done many very neat things at the Webster, New York hardware lab (or copier/printer things done at PARC). These have gone into products, at least in part because Xerox does have product groups in those areas. Research works at these companies, but it is certainly more difficult for them to get mileage out of software research than hardware. My belief is that this is directly proportional to the phenomena that it is hard for them to deal with most software products at all (apart from a couple of limited categories).

One final point is that somehow people seem to feel worse about great research sitting idle than the bottom line would indicate. Xerox's failure to capitalize on PARC is certainly a shame, but only from the perspective of what might have been—the direct loss of money was not the issue. From any financial or strategic standpoint, their "hobby" of supporting a world class computer research center pales in comparison to the $1 billion (it 1970 dollars!) they lost in the mainframe business. Even worse was Xerox's strategic failures which cost them many billions of dollars worth of market share in their core copier business. The hard bottom line is clear—to kill your company, you need a bad product strategy, and to waste a lot of money you need a bad product group. The actual cost of research never amounts to much in that context. This is not an apology for doing a poor job at technology transfer, but one should keep things in perspective.

The fact is that research *does* work at a lot of companies. When research fails it almost never is because of an intrinsic problem in research itself (i.e. the inability to think of something new). Instead, the research usually falls prey to problems that can be traced to general management issues—having the right goals, transitioning technology to company benefit etc.

This discussion sets the stage for the second answer to why research doesn't always work—you need to set your goals clearly up front. A great deal of the research which does not pan out is limited by things that could of or should have been determined up front:

> What are the deliverables to the company? Sometimes the goal is to advance the frontier of knowledge, sometimes it is to let management feel like they are good corporate citizens, and sometimes it is product related. You can't do all of them well at once, and sometimes they are mutually incompatible. It is important to recognize what research is trying to deliver to the rest of the company.
>
> Suppose that a project succeeds—will anyone care? Many research problems are plagued by the fact that the team doing the work is focussed 100% on whether and how they can achieve their immediate technical objective, and nobody is concerned about whether this objective would be a good one to achieve.
>
> What is the mechanism to transfer technology to products? At many companies the basic attitude is that they'll figure this step out once they get to it. Unfortunately, this is often too late, or the process of figuring it out is painful enough that projects die in the political fighting that ensues.

Our plan for setting up a research effort at Microsoft is to address these issues up front and build them into our system. This is not a magic formula for success—that does not exist for research any more than it does for product development. If the structure is right, the good news is that it is no more difficult, and indeed not much different from, product development and we can use our experience there to guide us.

3.1 Deliverables

The first question above is about deliverables—what kinds of things does the research group contribute and create? At the onset, we will eliminate status, image and philanthropy—those are not within the purview of this report. This leaves technology and its effect on the company. An initial question to ask is why us? What are the benefits that accrue uniquely to the people that undertake the research. There are many specifics, but the basic business benefits generically fall into three categories—you get something *early*, you get it *at all*, and *you* get it period. In more detail these are:

> Time advantage. A key reason to get involved in a certain class of project is that it will allow you to surprise the competition, or deny them the opportunity for a surprise. This is a weak form of access—instead of either having it or not, you have it early or late. Effectively using this time lead depends on having an efficient way

to take the deliverables from research and getting them into products.

Access to strategic technology. There is a common pattern which repeats over and over in our industry. Market and technology conditions evolve until suddenly a new technology is thrust into the limelight and it becomes a make or break issue. Outline fonts, RISC processors and handwriting recognition are all recent examples. In some cases there are many alternatives, but in others the only way to get access to the strategic technology is to do it yourself, because the people that developed it are content to use it as a weapon.

Ownership & education. Successful research creates intellectual property which is usually owned by the creator, and it also creates experience and know how within the organization. This is the thing you "get" directly out of research, but the big question is how uniquely you get it. Discovering a fundamental truth doesn't help if all you wind up owning is the copyright on the article that tells the world about it, of it others have time to invent alternatives. Just owning something is not much of a win unless you do so uniquely at least for some period of time. Know how in an organization is unique to that organization, but of course you need to have a way of capitalizing on it.

You shouldn't start research in an area unless there is a strong chance of getting a unique edge in one of these three ways. This sounds very basic, but most research done in industrial research labs does not qualify. This is not just bad luck—you can in most cases predict this long before starting. MCC and other research consortia inherently do not offer much to their members in this regard. It is not impossible for members to get any benefit, but it is tough because the difference between being a member and a non member is not sufficiently large, members compete with each other, the research that is done must by its nature be directed out far enough in advance of the market not to conflict with members work (which makes it hard to get a time advantage), and so forth. It is possible that a place doing really hot work could make the consortium approach work, but only if the work is compelling enough to overcome the barriers that the structure makes to the three points above.

The real question to ask ourselves is why should Microsoft do a particular research problem rather than letting others explore it? What is our value added, and why we will be able to turn the raw technology into a lasting benefit which is unique to us? There has to be a reason that we get a benefit, such as through time, strategy or ownership, which we would not or could not get if we simply let academic research or other companies pioneer the area. As an extreme case, this is why we probably won't spend a lot of time proving math theorems as an end unto themselves. No matter how fundamental the theorem, the transition between proving it and applying it is so great that a third party would be just as likely to capitalize on it as we would.

As another example, consider optical character recognition. It is clear that this will be very important going into the future, because we will need to bridge between the analog world and the new digital world which is being created. OCR is still a poor candidate for Microsoft to research

, because it is highly unlikely that we would get any of the advantages above:

> It has a generic interface. It is easy to treat OCR as a black box—bitmaps from a scanner go in one side and text (possibly with formatting) come out another. There is no unique advantage to incorporating this with other technology, setting an API standard, etc. At most we would own the code we wrote, some algorithms and potentially some patents (but see below).
>
> It is too old. There is little chance that we could get any fundamental patents in this area. We might make a technical breakthrough, but it is likely to be of an incremental nature—increasing the recognition rate from 97% to 99.9%, which is nice, but something to do in focussed product development —not research.
>
> It is replaceable. Because very few things get a dependency on the internal workings, somebody could come around tomorrow and replace it with a new and improved method which was utterly different. This risk is not a killer by itself, but you would have to be very certain about the time advantage you would get until this happened.

Before moving on, it should be noted that there may well be a worthwhile project in OCR lurking out there which manages to skirt these issues. The point is not condemning the field, but rather that you must confront these barriers. If somebody has an interesting new angle on the problem, then it may be well a good idea. This discussion just shows what the constraints are.

This covers the basic business issues, but leaves product and technology related benefits. These fall into several general categories. Suppose that we have done some terrific research project, and it has come to a conclusion. What are the sorts of things that the company might get out of it? Here is a list of the most important areas:

> Product vision and direction. Knowing what is possible, and what should be done to capitalize on it can be a key benefit. One of the primary benefits that Xerox PARC *could* have delivered was the vision of personal computing which developed from their work—in fact if you look at the Xerox research, a lot of it really fell into this category. The individual discoveries were good, but the overall vision was the biggest win—as the rest of the industry discovered.
>
> External API and format standards. Our business is driven largely on standards and one of the important contributions of new technology is creating externally visible features such as new programming models, APIs, data formats. It is possible for "black box" technology which is purely internal to be important, but it is more difficult for this to create a business opportunity for the owner or developer of the technology.
>
> Algorithms and know how. The most direct outcome of research is the fundamental technology which it comprises, which in computer science usually boils down to algorithms, design decisions and architectural issues.

Patents & intellectual property. An increasingly important part of the deliverables from research are patents. Unlike the other areas in this list, patents are unique in that they do not really require any sort of urgency in getting the technology to market—as long as you file early enough and get the patent granted you have a 17 year monopoly. The translation between a patent and bottom line benefit to the company is becoming more and more direct as companies turn to this mechanism for protecting technology.

Prototypes & code. The final deliverable in this list is the actual implementation of the research in code, which might be a prototype for a product or an actual component of a product.

This is basically in order of priority. This is not to say that the lower items like code are not important, because I would expect that each research project did in fact create code and a prototype at the very least. Nonetheless, this is not usually the major reason to do research—the code and prototype by themselves are not typically very important unless they also illustrate a new product vision, define a new programming model or draw on some of the other benefits. In certain circumstances the priorities can be utterly reversed—a patent, or the existence of a prototype to demo and show proof of concept can be crucial to business success in individual situations.

The list above may seem is may seem like an obvious enumeration of the possibilities. Like so many other "obvious" things, I believe that it is so straightforward that it is often overlooked. This serves as a kind of "check list" to evaluate a new project. Are we likely to create new algorithms? Will there be API and programming model impact?

Of course you can always be surprised in the course of investigating a topic, and one of the true joys of science is when this sort of serendipity strikes and yields unexpected benefits. This is a very powerful phenomena, and we want to encourage it by creating a collegial atmosphere for researchers to exchange ideas. I do not buy the concept that this somehow means you cannot or should not think up front about what the deliverables of a particular project are likely to be.

3.2 Will Anyone Care?

Given great deliverables, there is still a question of what the real impact of research will be. This is fundamentally a question for those *outside* the research group—the technical and strategic leadership of the company. The key thing is to be extremely focussed on getting synergy between the various research projects, and the general technical strategy of the company. The non-linear advantages that accrue when you have real synergy cannot be overstated.

The best way to implement this is by focusing the vision of the future on a couple of themes which are easy for the researchers to internalize, and identify with. You must also make sure that there are some people—essentially program managers—in the research group that can act as bridges to what the rest of the world and the rest of the company are doing. This

is discussed more below.

Another important point is to focus on problems where we are likely to get a big win. This again sounds obvious, but it too is often overlooked when research is planned. There are many risks associated with planning technology to be deployed in 2 to 5 years. The dynamics of our industry is such that many predictions fall by the wayside. Nevertheless, I believe that if you concentrate on the really big wins, and analyze the risks up front it is possible to come up with a research agenda which has a high probability of success. This is really no different than the existing problem of creating a long term vision and strategy for development.

3.3 Transition to Products

Once you have created some great technology, there remains the problem of effectively transferring it to the development organization. Failure to do this effectively is a primary reason that research work is ineffective at many companies.

There is no one magic formula to mastering this process—it must be managed throughout the life cycle of the research project. Some of the important factors are:

> High level strategic support is vital. The research group and the development groups must view each other as peers, and the best way to accomplish this is via the right support for the overall strategy within the company. This boils down to ensuring that the common themes and technical vision for the company are in fact shared and common to both. This process is largely "top down"—it requires the commitment of the technical and strategic leadership.
>
> Selecting the right research agenda is more than half the battle. The largest single technology transfer problem is that the technology is off target and nobody wants it or needs it in their product. This is a very vital point—no amount of technology transition "process" can help the wrong technology at the wrong time. The criteria listed above early in the process should solve a lot of the problem.
>
> Proper program management keeps the agenda relevant. The process of tracking the rest of the world, and measuring research goals against our strategic needs is not just an up front thing, but has to be maintained throughout the process. This is the job of program managers in the research groups
>
> Communication with product groups is essential. This is another responsibility of the program manager in each research area. The development groups are their direct customers and it is important for the research group to maintain a direct channel to the program managers in the product areas. They would also be responsible for organizing retreats and brainstorming sessions which bring product people in contact with the research.

The basic model is that development groups would consider the research group to be a

source of technology similar to an outside company licensing technology. They could get consulting time and so forth, but the researchers would not be expected to move directly to product development. We could have people in product groups transfer in to research and then move back with expertise, but it is not a matter of policy to move the research group wholesale into development. On an individual basis such transfers could occur of course—the issue is that it is hard to sell people on coming to the research group if it is viewed as a transitory way station which as a matter of set policy will convert people into developers as soon as their research is applicable.

We have considered (and tried to a limited extent) other methods, including the "pass through" model where people from development move into research, then back out to development to productize it. There are a number of subtle issues that have to be watched in order to make this successful. Excellent developers can make poor researchers, and visa versa. The notion of moving people with projects is nice, but it is no panacea. In the long run, a pass through structure might be a valuable thing to set up, but the primary goal at first is to build up a permanent research group which can have its own identity. Once that is sufficiently established, it will be able to absorb people in from, and out to development without changing the basic focus of the research group.

4. Structure and Organization

The basic idea is that there would be a unified research group which would report in to ATBD, and which would have sub-groups or labs which focus on particular topics. The pros and cons of this approach, and the details of how to implement it are discussed below.

4.1 Why Have a Research Group?

A question which comes up at the onset is why have a focal point for research in the company? Instead, you could distribute experts throughout product development groups which were most relevant to their work.

I do not believe that this method would be successful from a variety of standpoints:

> The best research people will not come under those terms. There are people working in universities, or at places like Bell Labs, Xerox PARC, IBM Watson Research Center, DEC's Systems Research Center who are very smart, dedicated, and interested in problems that we want to solve, but who would simply never consider going to a product group. In part this is because of the reputation that product development has in some companies (particularly the companies listed above) which colors their perspectives, and in part because there really is a difference. Either way, it is a practical barrier to hiring a lot of talented people if you insist in putting them directly in a product group. Of course by the same token there are people who want to directly be responsible for shipping products, and they would probably look askance at working in research.
>
> Can't create the right atmosphere. Culture and atmosphere are hard to pin down up front, but very apparent in practice. A product group which is working on a

deadline and is out to nuke the competition is just a lot different than a research group—no matter how driven and focussed the researchers are.

Synergy between research efforts is hard to obtain. This is a crucial thing to attain, especially if we want to focus our efforts on a common theme, but it is very hard to do if the research activities if they are scattered across the company. Like atmosphere, this is hard to quantify, but it is a very real effect.

Product groups are not equipped for this. Everybody professes an interest in the future, working on new technology and so forth, but frankly speaking not everybody is good at this, or even comfortable with it once they actually get to work. Our developers are smart, that is not the question at all, but they, and their management have been selected and tuned for a different set of goals.

One could make the argument that precisely for these reasons, it makes sense to try to do it—i.e. to change the attitudes and increase the innovation in product groups. Unfortunately this is both difficult to do, and possibly undesirable. Scattering a few visionaries in the midst of non-believers who are absorbed by their product commitments is not the way to change the organization. Also, it is not clear that you want product groups to be much different than they are today—their job is to integrate and implement technology to build great products, and they are good at this. The job of research is to take unsolved problems and convert them into a tangible enough form that product groups can absorb them into products. Practically speaking, I believe that the best way to simulate innovation in product groups by presenting them with concrete technology they could apply in an interesting way.

Of course product groups should be encouraged to do as much advanced work as they want to do. The point here is not to limit what product groups are able to do, but rather that it makes sense for the company to have a unified research group.

4.2 Proposed Structure

The basic structure is that there would be a research director in charge of managing the overall research effort. This person is similar to the chairman of a computer science department in a university (and in fact that is a potential place to look for recruits)—someone who has sufficient technical stature to be respected by researchers and technical people, but who is also able to be effective at managing and motivating people. It is important that the research director be able to recruit effectively for the other positions and it is vital that the director be able to instill the right team spirit and atmosphere for the team.

This is not to say that the research director shouldn't think. It would be very difficult to attract the right kind of researchers unless the research director has substantial techincal credentials. The department chairman analogy mentioned above is only partially correct. A real university department chairman does not have much influence over the research going on in the department, but the research director in our group will have this influence and involvement.

At the same time, we must remember that in the first year or two this is very much going

to be a "start up" operation—to the same degree as a new company. A team has to be hired, a reputation has to be created from scratch, projects have to be defined and gotten underway etc. The task of leading this peculiar kind of start up is not a part time effort. A start up business usually cannot afford to have part time management once they get down to business, and neither can we. We certainly should attempt to hire a smart director who has a substantial reputation as a researcher and who can continue this in the future, but we need to make sure he or she sees the organization as their primary responsibility. We do not want somebody who will retreat into their own research and just have the director title as a mark of seniority or prestige. The realistic expectation is that for the first year or so there will be *plenty* to do in orchestrating the bootstrap process and that would be the director's primary responsibility.

There is a theory that says you really need two directors—a smart visionary sort of person and a "Mr Inside" who does the management and administration. The person described above for the director is Mr Inside, and this is our most important internal need. Depending on the actual people we encounter, it might make sense to have this kind of dual structure.

Underneath the research director would be a number of research groups. Each of these would have a manager who also has research duties, perhaps as the technical leader of the group or perhaps not, depending on the group. The groups would be small enough that the managerial duty should not be too cumbersome. There would also be a technical lead and a program manager for each group. The program manager position may seem unusual from an academic standpoint, but as mentioned above, the program manager is very important for coordinating with product groups, keeping abreast of the competition and managing the research agenda. I believe that this is a very important way to achieving tangible business results, and melding the research group with the Microsoft community as a whole.

The total number of people in each group would vary from 5 to 10, depending on the area, the scope of the projects etc. Note that group does imply one project—there may be more than one project or sub project—the group would be in a given technology area such as Natural Language etc. It is possible that we would have a couple "lone wolf" researchers that would basically be working on their own, or with one additional person, depending on the situation.

The following diagram illustrates this organizational structure:

This discussion uses titles and names which are similar to those used in the rest of Microsoft. In practice we may want to use somewhat different nomenclature. As an example, we might want to call each of the research groups a "laboratory," so a group doing imaging research would be the Microsoft Imaging Lab. The precise name of the overall research group is another open issue—it is a "group," a "unit" (as in research units), a "division," a "lab"... There are lots of possible names and this needs to be thought through. At a personal level, we probably should keep the "program manager" title fixed, so that other program managers here will recognize them as one of their own, but the other titles may not be optimal.

5. Headcount & Resources

The clear message that we have gotten from people in the research community is the most significant factor in attracting people is that we show sufficient *commitment* to research. This comes in several forms:

> Commitment to invest sufficient resource in research to reach critical mass. It is not possible to attract people for either the research director position or even the individual researcher positions unless there is a plan to be serious. It is universally believed that you cannot be serious with a handful of people—the atmosphere and synergy that research thrives on requires a certain size.
>
> Commitment to fund research so that people don't have to beg or have their projects cut. This is an extremely common fear among research people. Research groups are regularly eliminated (as an example, last year Olivetti fired its whole Advanced Technology Center with just two weeks notice). Nobody wants it to happen to them, and everybody knows somebody who has had it happen to them, so the concern is very immediate.
>
> Commitment to have vision and be open to put research into products. Finally, there is considerable frustration at many top research places because their product groups have very little vision and are usually not interested in what the researchers have done. It sounds amazing, but just appearing to have an open mind on this area wins the hearts of many of the researchers. They want to effect products, and many places won't let them. This is a very important way for us to attract really great, and practically minded, people. PARC, Bell Labs and Watson Research center may have a lot of credibility as places to think, but Microsoft has terrific credibility for an ability to ship products which change people's lives.

These commitment concerns are right at the top of everybody's list. You cannot talk about the position without them coming up—it is more important than compensation or any other issue. The current state in the world at large is that researchers are very cautious because they have been burned, or heard of friends getting burned, so they really think in terms of up front commitment.

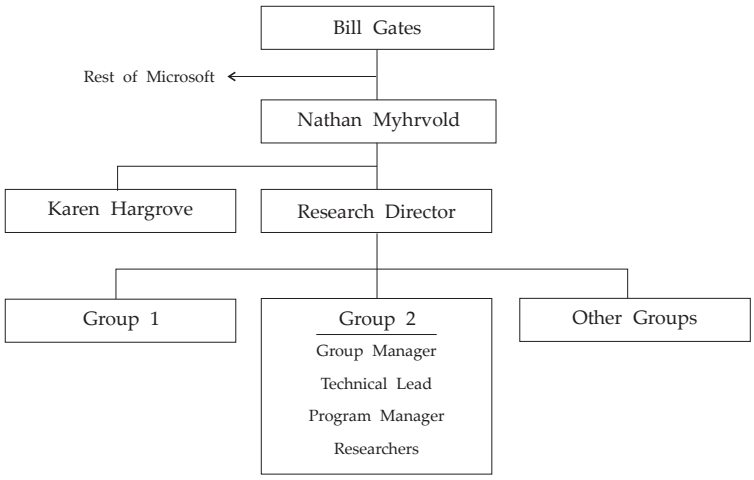
This is an interesting situation. If we show firm commitment on these points (mainly by looking them in the eye and saying so), then we instantly stand head and shoulders above other research establishments, and can hire the best people, or at least have a very good shot at it. If we are wishy washy or do not make a sufficient up front commitment, we are in last place because we will appear to not know what we are doing—i.e. a bunch of hackers writing for toy PCs that think they need research, but don't know what to do about it.

The first area is critical mass—having enough people working in research areas that we can get serious work done. The magic number that most people seem to quote is a minimum of 50 people. Note that we do not have to hire that many people at once—everybody understands quality control, and nobody would want to grow too fast. Nevertheless, you have to have a

stated goal of reaching 50 people within two years, or you seem like a dilettante.

In this scenario, the first year the headcount for research would be 30, and the second year would be 50, with some reasonable (but smaller) growth in the third year—say 60. These would all be new heads, in addition to the present ATBD headcount.

Note that even in the third year with 60 people, the total investment in research is pitifully



small for a company of our size (especially if you project what our size will be in three years). If we went by the same criteria as most Japanese companies (Sanyo, Hitachi, Ricoh, Sony etc) we would have over 300 people doing research—and that is going by our size today. I am not advocating numbers for the sake of it—but we have to remember that outsiders *will* judge us by these standards to some extent. They will also consider the absolute number of people it takes to get several reasonable research efforts going as a viable "critical mass".

It is not clear whether we will actually hire all of the 30 people in the first year, but it is quite possible. We have already been contacted by research groups at HP Labs, IBM, and the IRIS group at Brown who want to leave en masse—in each of these cases we could pick up 3—6 people as a team in a short period of time.

I am perfectly willing to accept a contingency statement whereby the heads are not actually released until we meet milestones, such as hiring the research director etc.

6. Recruiting

Each of the research teams should be lead by world class people—there is really no excuse for settling for second best. I believe that we can get people to do the best work in the world in the areas we choose to enter.

The initial recruiting plan is as follows:

Hire Gordon Bell as an advisor to the research group. The key initial task is to get our plans straight and then attract the right people, and Gordon would appear ideal for this. He would probably not be available or best used to actually lead the group, but he would be ideal for finding the research director, as well as many of the technical leaders for each area.

Consider having an advisory board. We have long discussed having a scientific advisory board, and it especially makes sense for the research group. One of the near term benefits in setting such a board up is to get the board members to assist in referring people, and to enhance our reputation. The board could include people like Gordon Bell, John Hennessy and Doug Lenat—very well known people who have had enough contact with us to be easy to recruit to the board. We would want to try to keep it small at first so that it was not a big overhead in and of itself but it could be a net positive even in the short term due to referrals.

Pursue the director first. The highest priority is to go after the research director, so that he or she will be on board to help with the rest of the recruiting, and to manage the group as it grows.

Be open to opportunities. Although the research director is the top priority, special opportunities may arise which warrant immediate action. The IBM natural language people are an example—we have to strike while the iron is hot in order to get them. Hiring the core team of three people can occur before the research director is hired, and in fact should lend some credibility to the effort because of their reputation.

Go after experts once we have firmed up the mission in each category. We need to have a reasonable idea of the research agenda before going out and hiring people. Although the staffing level would be committed to 30 and then 50, we would not just open 30 reqs the first day—we want to be careful to match the research missions with the right groups, and get the right people for them.

Target specific experts once the research area is identified. Part of the process of investigating an area for potential selection as an Microsoft research project would be to list the best people doing the work, and to directly talk to them about their work. This is the best way to get the information, and it is also a good entrée for recruiting. In the longer term we would rely on a variety of recruiting programs to attract new Ph.D. students and established researchers. An example is the "visiting scientist" position where university professors could come work at MS for a year. This is typical practice in universities and some research centers, and in Microsoft terms you can think of it as an advanced version of the summer intern program! Besides the benefit of the work that they do, this helps establish a relationship for getting their graduate students in the future.

7. Research Agenda

The primary purpose of this document is to lay out the plan for building the group rather than listing all of the research that will be done. *The discussion below is simply meant to illustrate the kinds of projects that are envisioned.*

Although there will be groups in a variety of different technology areas, it is vital to focus our efforts toward some common themes which are shared by all of the groups. This serves as a way to communicate our goals to everyone on the team, and try to channel spontaneous creativity in the right direction. Example research themes are:

> Information at your fingertips. This is to be interpreted in the broadest sense—making the personal computer into an information access and reading tool rather than just an authoring tool. The implications of this campaign go well beyond our present set of projects in this area, and will provide a lot of opportunity for research.
>
> The digital world. The world is going digital, and this creates enormous opportunities for integrating devices, services, and even whole industries which have been quite distinct in their analog manifestations. All current means of delivering information are suddenly going to be on common ground. The center of this cyclone is the personal computer and the software inside it. PCs are where this information will be created, and they are the vehicle though which they will be delivered. There is a great deal of research to be done in putting this together—both in how you manipulate, store and distribute the data, as well as the IAYF issues which focus on how an individual copes with it.
>
> Creating the digital office and home of the future. There are many interesting problems in computer science, but we want to focus on those that will become part of the mainstream of personal computing—the things which will help office workers and the ordinary "man in the street" who will increasingly rely on computing technology as a vital part of their lives.

If these sound redundant, it is by design. They are just different ways of looking at the same thing—how the personal computer will evolve between now and the mid 1990s.

In addition to the general themes, we should strive to have as much in common between the projects as we can. The strategic environment for all work will be Windows (or more precisely, the target environment is one that supports the Win 32 API), and most of the work will probably be done on top of NT. Some of the work may wind up in future versions of Windows, some may be in applications on top of Windows, but in any event, Windows is the core. In addition to being the target, Windows would also be the working development environment. This gives us good product feedback, and it also lets us develop tools that can be shared across the group.

I would also like to see us focus on as few implementation languages as possible—ideally just one, which probably means C++ (with ordinary C as an acceptable subset). Getting the AI people to use C++ instead of Lisp may or may not be feasible (I am actually quite optimistic),

but it would be nice if we could share as many tools etc. as possible. This kind of detail may sound like a nit, but it is one of many ways that you build synergy in the effort. Historically, the most successful research groups have often been very hard core about this—at Bell Labs essentially everybody uses C (and UNIX). Xerox PARC changed its mind on languages several times(SmallTalk, Mesa, Interlisp-D), but at any point in time 90% of the work was in one of them. Just about everything done at IBM Watson Research center is in PL.8 and DEC SRC uses Modula 3 exclusively. Of course each of these places invented the languages they fell in love with, and I see no reason for us to do that—in the near term[1] at least!

Another point of philosophy is that we should have *users* of the technology within the research group. That is one of the motivations behind the two research projects listed below to explore new application categories. They will not be required to use technology developed in other research groups, but there will be ample opportunity to do so, and synergy between the various groups will be encouraged. It will also be typical for research groups to make their own prototype applications or demos. We do not want to fall into the trap of designing theoretical systems years before application writers get to see them. This is precisely what we did in the case of Windows and Presentation Manager—each[2] were designed in what amounted to an "open loop" fashion—so this is a very real concern. This phenomenon is occurring today in with IAYF —most of the thinking that is being done is in the systems components and not enough at the application level. We want to make a conscious effort to organize the research agenda so that application level thinking is fully up to speed with system issues—and ideally out in front of them.

One reason to do this is that there is an enormous amount to be learned from having the entire "food chain" interact—the application people can beat up the add-on library people who can complain to the kernel people who will rant at the people supporting exotic devices and so forth. The feedback you get is often invaluable. When we invent a new programming model, thousands of ISVs will have to live with it for years, so we want to get as much review, from all of them as possible done up front.

7.1 Research Activity Decisions

Themes set the tone and get people aware of the basic thrust of the research, but there is still the process of deciding what projects to do, and what areas we should investigate. The general methodology for doing this in the start up phase of the research group will be:

> Do a lot of homework. It is important to put a lot of effort into checking the area out up front. This includes going through the list of criteria discussed earlier in this memo, as well as reviewing the technical issues.
>
> Identify existing work and experts. Depending on the field there may be a little or a lot of relevant research being done at other places.
>
> Prepare a proposal. This will outline the general direction for research, the benefits and who we should attract.

Recruit team. Note that it most cases we will expect to rely on outside experts rather than going inside the company, although that is not out of the question, especially for programmers and program managers.

Kick off detailed planning with a retreat. Once we have the experts on board, the typical way to initiate the actual planning would be with a retreat that involved key technical people across Microsoft, the research team and some advisory board members.

Do not succumb to process or bureaucracy. The list above may sound like there is a very formal process by which we decide to do a project. If that really occurs, then we are doomed. There has to be the right sort of dynamic involved in balancing common sense (which is mainly what this is) with formality. It is silly to open a major lab without a little homework, but if you require a ton of paper before you follow up on a spontaneous idea, then you kill creativity. This tradeoff has to be properly balanced.

In the long term one would expect that ideas and proposals come from a variety of sources—other research projects, requests from development and so forth.

Endnotes

1 Using "rich source" to view the code in a very different manner might be interesting.

2 Especially the User component, the input model, and the window manager.

A Short History of Microsoft Research

Dan Ling and Jack Breese

In September of 1991 Rick Rashid arrived at Microsoft to officially found Microsoft Research. This was the culmination of months of recruiting effort on the part of Nathan Myhrvold, at the time the head of The Advanced Technology and Business Development division at Microsoft, and Bill Gates. Rick was one of the stars in computer science systems research, the director of the Mach operating system project, and thus a very successful professor of computer science at Carnegie Mellon University, one of the premier departments in the country. So coming to Microsoft, which at the time had no track record of doing research, was a risky decision for Rick. During this period Rick came to be known as 'Dr. No' to Bill and Nathan, given the number of times he declined their offer. However, fortunately for Microsoft Research, Rick finally accepted the offer and arrived at Microsoft.

The planning for MSR, however, began well before that date. Nathan Myhrvold, in discussions with Bill Gates and senior Microsoft leadership, wrote a key memo proposing the new research group. The memo made the case that Microsoft, and by extension the PC industry, could no longer simply depend on innovations from the earlier mainframe and minicomputer eras. The increasing power of the microprocessor, as well as new developments in hardware and communications technology, would drive completely different usage scenarios never before contemplated. To solve those hard technical problems Microsoft would need to do basic research.

What is often not appreciated today is that Microsoft was quite a small company in 1990-1991 when these discussions were taking place. In 1990 Microsoft had just crossed the $1B revenue mark. Windows 3.0 had been released but not the hugely successful Windows 3.1. The various Microsoft Office applications were not yet the clear market leaders. Microsoft was a successful and growing company but certainly nothing like the powerful force it became in the late 1990s. So it was with remarkable foresight and courage that the company decided to start a research group at this time.

Objectives and Principles

During the 1990s MSR was occasionally referred to as "Bill's Labs" in reference to the still (at that time) dominant and influential Bell Labs. In some ways, MSR did emulate the Bell Labs model—it performed both theoretical and applied research and was sponsored by a major industrial player. The research arm was organizationally distinct, reporting to corporate management as opposed to residing in an operating or engineering division. The most important similarity was that both labs were based on making a long-term commitment to basic research as part of a commercial entity. The research was not exclusively tied to the short term (say less than a 5 year time horizon), but rather the portfolio included a mix of projects of both long-term and short-term promise, not all with obvious potential commercial applicability. Critically, at both institutions, there was a commitment to furthering the state of knowledge in the relevant fields of endeavor (e.g., physics at Bell Labs, computer science at MSR) without obvious direct benefit to the corporation.

Rashid, along with "co-founders" Myhrvold and Gates, achieved this mix of short/long and applied/basic research through a set of operational principles and strategies that were applied from the very beginning and continue to this day. One of these basic principles at MSR is that research programs and priorities are essentially driven by the researchers. Of course this can only work when the research organization is staffed appropriately and hence at MSR, as in the rest of Microsoft, there was a tremendous emphasis on hiring the right people. MSR looked for researchers who were passionate about seeing their ideas put into practice and, via the huge reach of Microsoft products, literally changing the world. Management would occasionally highlight challenges facing the company or industry at the time, but there was very little top-down direction. Researchers at MSR are expected to be world-class experts in their field and to be self-motivated as far as working to see their inventions and innovations incorporated into Microsoft products. Rick set up a management system and culture that fostered this mode of operations, starting first with recruiting research teams that acted as small entrepreneurial "startups" within the larger Microsoft R&D system. Often this meant recruiting a team that had worked together at a previous institution. Rick's messaging to those groups was not typically specific projects or tactics but rather "do the right thing," with respect to choice of topics, hiring, research expenses, and mix of short and long term research projects within a group.

In the early years, locating the research group (in building 9) on the main Microsoft campus close to the product groups was critical to providing opportunities for the researchers to meet frequently with engineers and leaders. This allowed researchers, over time, to understand the product strategy and build a research agenda around future technical challenges. Building an alignment in research and product strategies over a number of years results in a much more relevant and potentially impactful research program, based on rich and multiple channels for feedback. In addition, these regular interactions were necessary to build the personal connections and trust on both sides to enable shared technical commitments and adoption of research innovations.

Another key principle has been the importance of scholarship and research excellence. To quote Rick, "Our researchers are here to push ahead the state of the art in computer science. When we have great ideas that work, we strive to move those ideas and technologies into Microsoft products as rapidly as possible." This evidences itself through an emphasis on publications at competitive conferences and journals, generation of intellectual property, and broad engagement with the global research community, through participation on editorial boards, conference program committees, etc. At some level this acts as a quality assurance function, as management cannot judge the overall quality of research in a vacuum. And ultimately, in the highly competitive and dynamic software industry, it is critical to transfer technology that is innovative, defensible, and capable of supporting a sustainable competitive advantage.

Central to Rick's leadership at MSR has been consistency with respect to the principles just outlined. This is relatively rare in corporate research. We have seen tremendous changes in emphasis and funding levels at other large industrial research organizations, such as Bell Labs, XEROX Parc and IBM Research. This consistency is a testament not only to the extraordinary performance of Microsoft Corporation and Microsoft Research over the last 20 years, but also to Rick's ability to demonstrate to Microsoft management the benefits of a stable and substantial research effort.

Growth

Dan Ling's personal history started in early 1992 when he first visited MSR. At the time, Dan was working at the IBM TJ Watson Research Laboratory. Because Dan and Rick had been friends since their freshman days at Stanford, Dan had become intrigued when Rick asked him to visit Microsoft after he had arrived to found MSR.

At the time of Dan's first visit there were only half a dozen researchers at the lab including the core members of the Natural Language Group anchored by George Heidorn, Karen Jensen and Steve Richardson, who had arrived from IBM about a year earlier. Some early members of the Systems group were arriving, mostly new PhD graduates from CMU. In addition, some early work on a new software development tool later known as the Basic Block Tool (BBT) had started. This tool optimized the working-set size (use of real memory) of paged applications. This tool would later have significant impact on Microsoft during the Windows 95 timeframe when systems transitioned to 32-bit code but real memories of existing PCs were quite limited. By allowing significantly improved performance of applications such as Office 95 on PCs with limited memory, BBT proved to be one of the earliest MSR contributions to Microsoft products.

Even though the group was quite small at the time, the promise of the lab coupled with the overall enthusiasm at the company made the opportunity irresistible.

After Dan started at Microsoft he was asked to start building a group in Human Computer Interaction and Computer Graphics. We were guided by the philosophy that it was important to have a critical mass of activity and talent in each of the areas we wanted to pursue. And

to do so, it was critical to seed groups with distinguished senior leaders who could provide vision and guidance and also attract new talent.

Jack Breese, along with colleagues David Heckerman and Eric Horvitz, first visited MSR in the summer of 1992 for what they thought was to be a research talk and visit. The teams mentioned above plus Charles Simonyi's Intentional Programming group and a team headed by Daniel Weise working on program analysis were also in residence. It turned out to be what they later learned was a "sell" interview, including luxury accommodations and dinners, orchestrated by Nathan Myhrvold. It took several months and the dogged perseverance of Nathan to finally convince them all to leave their startup and research careers in Silicon Valley and move to Redmond. The newly formed "Decision Theory Group" was focused on building aspects of intelligence and improved decision making into Microsoft products, and in due time had a mix of researchers and software developers making up the team.

It is important to remember that Microsoft had no track record and little credibility at that time in the research community. Finding a recognized leader in the field and attracting them to MSR was not at all an easy thing for a small unknown research lab. One of the memorable events during this early period was our effort to attract Jim Kajiya in 1994 to lead the computer graphics efforts. Brian Guenter and Dan had identified Jim as one of the leading researchers in the field and invited Jim to visit MSR for a week in order to convince him of the excitement and promise of the new laboratory. Because the group was so small at the time, in order to fill up the agenda for a week we scheduled meetings not only with the researchers, but also with developers in the larger Advanced Technology group and in the various business groups. In addition Jim met with various senior managers. We were very fortunate that Jim decided to come to MSR where he helped nucleate the graphics group which over time became one of the premier research groups in the world. As Nathan Myhrvold recalls this hiring, Jim was one of the first researchers recruited to MSR on the basis of it being a great place to be in terms of colleagues and environment at the time of hiring, as opposed to previous recruitments being mostly based on the future potential of MSR.

The years between 1995 and 2000 formed a period of very rapid expansion for Microsoft Research. The laboratory in Redmond grew over threefold. One of the early research priorities was the idea of computers that could interact with humans beyond simple GUI interfaces, sparking work towards computers that can speak, see, and think, interacting with people on human terms. This led to the creation of some of the earliest groups in the lab—natural language processing, computer graphics, computer vision, decision theory and AI, human-computer interaction and speech recognition. In addition there were also groups working on systems and networking, databases, programming languages and software development tools which dated back to the very early days of the lab. New groups were also formed in data mining, cryptography and security, streaming media, devices and computer-mediated collaboration to address market needs and new opportunities for the company.

During this period the laboratory also began to hire research software development engineers to work alongside the researchers. We had a strong belief that it is important to build software artifacts of considerable size and complexity to convincingly demonstrate new research ideas.

In the process of building this software, functionality could be refined, algorithmic and programming bugs fixed, and performance issues addressed. These added steps would make it easier and faster for the development groups to incorporate research advances into Microsoft products. It also had a role in easing integration and communication with the product groups, in that experienced developers typically had contacts, as well as intimate knowledge of Microsoft development and release processes. All of this proved invaluable in creating true partnerships between researchers and product teams.

MSR also started to grow outside of the Redmond corporate campus, out of the recognition that not all of the researchers that Microsoft would want to engage would or could move to Redmond. This realization started with the creation of a small group headed by Turing Award winner Jim Gray in San Francisco in 1995.

In addition, it became apparent that as a growing international enterprise, Microsoft and Microsoft Research needed to have research perspectives and insights from non-US cultures, from both Asia and Europe. In 1997, the first international lab was created on the Cambridge University campus and headed by Roger Needham. After a number of trips to China by Dan Ling starting in 1995, it became apparent that a combination of factors made China an ideal location for a second international research laboratory. This included the growth of the Chinese economy, the flourishing level of entrepreneurship, the rapid improvement in science and engineering education, and, perhaps most importantly, the enthusiasm of the Chinese students in the area of computer science. MSR Asia, located in Beijing, was founded in 1998. One of the major areas of emphasis for the lab was input methods and processing of Asian languages. The vision we had for MSR Asia is that it conduct research with the same degree of independence as the other labs and with the goal of having the same quality and impact, whether measured in terms of recognition by the academic research community or contributions to Microsoft products. This level of expectation was quite beyond what most multinationals had for their research facilities in Asia, and was fully realized over the next decade. The emphasis on outstanding quality allowed MSR Asia to hire top-tier researchers including new PhDs from local universities, more experienced local researchers, and returning professionals from abroad; all of whom contributed to the laboratory's success.

The growth of MSR reflected the growth of Microsoft at large; between 1995 and 2000, Microsoft revenue almost quadrupled. Although MSR was given the budget and headcount to grow rapidly, growth always adhered to some fundamental principles. Hiring was driven by the opportunity to bring to MSR truly outstanding researchers who had a strong technical vision and a passion for seeing research translated into impact, not just someone to fill a role or meet a job description. Most of the new hires were new PhDs but, in order to start initiatives in new areas, it was also important to occasionally bring in more senior people as well. It was also important to assure that any research area have critical mass so that researchers could collaborate and share ideas with peers in their area.

As we entered the 21st century, additional laboratories were formed. In 2001 the Silicon Valley lab was created with a focus on distributed systems. This lab came to include the group in San Francisco as well. In 2005, the MSR India lab was formed in Bangalore. We realized there was

a tremendous opportunity in India to conduct outstanding research and thereby participate in a rapidly growing IT industry with passionate students and strong entrepreneurial spirit. Finally in 2008, MSR New England was formed in Cambridge, MA to engage more strongly with East Coast universities as well as new Microsoft development initiatives in the area. Although MSR has grown to over 1000 researchers in 6 labs, the principles established at the beginning are still in place.

Some Fundamental Contributions

Over the last twenty years, the research staff of every MSR laboratory have made many significant contributions to computer science. One only has to look at leading international conferences and workshops in a variety of fields where the prominence of MSR researchers among the lists of authors (including many with best paper awards), presenters, panel members and program committee members shows the impact that MSR has had.

We would like to point out, however, several particular areas where MSR has made fundamental contributions that have shaped and changed the research direction and deeply influenced colleagues around the world.

Computer Graphics and Vision

One of the major breakthroughs in computer science that Microsoft Research helped nucleate is the merger between Computer Graphics and Computer Vision. These two fields, which have both been very active since the mid-1970s, had traditionally published in separate venues and not built on each other's work.

Computer graphics is the study how to render and animate scenes on a computer from given 3D geometric and appearance models. Conversely, computer vision develops techniques for analyzing 2D images in order to recognize their contents and build 3D models. (A third, related field, image processing, aims to improve the quality of images through analysis and re-synthesis.) Until the mid-1990s, these fields, while sharing a common mathematical framework, largely operated in isolation.

Microsoft Research at that time had assembled a world-class cadre of researchers in both fields. What emerged from this confluence was the field of *image-based modeling and rendering*, in which computer vision algorithms are applied to real-world scenes in order to recover approximate shape and appearance models, and computer graphics algorithms use these models, along with the original images, to create photorealistic interactive animations and virtual tours. Much of this work appeared at the leading graphics and vision conferences, such as SIGGRAPH and CVPR. Its impact on the graphics and vision industries includes widespread use in visual effects as well as the creation and navigation of rich 3D mapping experiences, such as street-level tours.

Around the same time, the idea of analyzing multiple images and re-synthesizing better outputs started being applied to everyday photography. An early example of this, pioneered at MSR, was the creation of seamless panoramic image mosaics using sophisticated alignment and

blending, i.e., "stitching," techniques. While these could be used to generate interactive image-based rendering experiences, they could also be used to generate beautiful photographs. This field became known as *computational photography*, since large amounts of computer analysis and synthesis were being inserted into the traditional optics-based photography process to create beautiful, previously unrealizable images.

Additional examples of computational photography developed at MSR include high dynamic range photography, where multiple exposures are seamlessly fused to capture very dark and light regions, and the merging of flash and non-flash images to create well-exposed photographs in low-light conditions. They also include removing blur and noise from low-light images and merging multiple photos to get the best smile on everyone or for more creative "fakery." All of this research has led to the creation of a new field, with its own annual conference (International Conference on Computational Photography).

Formal Methods in Software Development

As a second example, researchers in MSR have led the way in applying formal methods to improve the quality of real-world software by finding bugs. While the use of formal methods in software development has a very long history, this effort followed a very different path than the classic program of verification research, which attempted to prove the total correctness of a program, a far more challenging goal than simply identifying errors in programs. This change in objective, however, was fundamental to the success of MSR in producing highly-influential new research results and practical tools that aided developers. The most prominent example is the SLAM software model-checking research, which combined ideas from formal methods, model checking, and program analysis with original and innovative insights to produce a technique for verifying the temporal safety of software. This research enabled the development of SDV, a defect-detection tool for device drivers, which Microsoft has distributed and supported for almost a decade. SDV was widely accepted by driver writers and improved the quality of this crucial, error-prone operating system component. Equally important, SLAM opened a fruitful dialog among the programming language, software engineering, and formal methods communities that benefited all and led to many innovations in program analysis, defect detection, and testing tools and improved software development in general.

Satisfiability-Modulo-Theories (SMT) solvers were a critical part of the SLAM tool, which used them to discharge sets of logical constraints for the automated abstraction of C programs, the symbolic execution of program paths, and the refinement of Boolean program abstractions. Soon after this work, MSR began development of a sequence of theorem provers, eventually culminating in Z3, an SMT-based theorem prover whose high performance has facilitated its use in a wide variety of defect detection and testing tools, both within Microsoft and in the larger research community. New algorithms, faster processors, and larger memories facilitated a broad shift from compiler-based techniques, such as data-flow analysis, to more precise approaches based on SAT-solving and theorem proving.

From the foundations of SMT solving, work in symbolic methods in MSR branched out to consider other problems and domains, including fixed-point computation (to support program analysis

tools), interpolation (to support automated program abstraction and refinement), automata (to support reasoning about programs that manipulate strings), and systems of non-linear equations (to support reasoning about mathematical functions).

Although theorem proving figured prominently in early work on program verification, its largest current application inside Microsoft is in program testing, where MSR has pioneered hybrid static-dynamic analysis and built the SAGE tool, which identified a significant fraction of input-dependent ("fuzz") bugs that provide pathways for potential attacks in a variety of Microsoft products.

Probabilistic Perspectives in Artificial Intelligence

During the 1990s, MSR was a leader in an emerging intellectual and academic perspective on performing machine intelligence research. The basic idea is that probability theory and decision theory are important tools in performing and assessing perceptual AI tasks, specifically in such fields as speech recognition, natural language processing, search, natural user interfaces, AI planning, and computer vision. From the very beginning, across numerous fields that might be characterized as perceptual artificial intelligence and intelligent systems, MSR amassed a set of researchers who shared a perspective that notions such as Bayesian probability, utility theory, Markov decision processes, and statistical inference were useful tools in the deployment and development of intelligent systems

MSR was unique at this time in having a critical mass of such researchers, and this has had a tremendous impact on the field. MSR had serious research efforts which adapted these probabilistic methods in such areas as adaptive user interfaces, search, machine learning, data mining, speech recognition, computer vision, and natural language processing. In the late 1980s and early 1990s AI research, in fields such as natural language processing and AI planning (among others), the preponderance of representations and inference techniques were logical in nature with no provision for explicit management or acknowledgement of uncertainty. And if they did deal with uncertainty, typically, non-probabilistic or ad-hoc methods were utilized. While it is not the case that all or even most of the research with probabilistic/decision theoretic perspectives was performed at MSR, it is the case that the success the MSR groups had in both academic research and real-world applications in Microsoft products was an exemplar for researchers in the field. Currently these approaches are nearly universally accepted and acknowledged and are considered foundational for much of the artificial intelligence and intelligent systems research performed today.

Summing Up

Over the last 20 years, Microsoft Research, under the leadership of Rick Rashid, has had a broad and deep impact on the field of computer science. The field itself has expanded dramatically into new areas such as computational markets, social computing, mobile computing, and cloud-based services; areas in which MSR has grown expertise and adapted in kind. While not emphasized in this article, MSR has simultaneously made significant contributions to Microsoft operating system, application, and online services product lines.

An exciting recent development, representing a close partnership between MSR and the Microsoft Xbox team, is the widespread success of the Kinect motion-sensing Xbox video game controller and interface. The adoption of this technology into general-purpose computing, coupled with advances in speech recognition, language understanding and other intelligent user interface methods will perhaps herald a new era of human-computer interaction, and fulfill an early MSR mission to develop computers that can see, speak, and learn. Only time will tell the true place of MSR in the history of computing, but now, as in 1991, we are only in the beginning of the impact of information technology on society.

A Letter

Linda Stone

One of Nathan Myhrvold's best decisions was to pursue and recruit Rick Rashid. I wonder if he could patent any aspect of this? Maybe aspects of his oddball recruiting practices—many of which he attributes to Charles Simonyi. In the end, Microsoft was lucky enough to get and keep Rick, and Nathan is now Lord and Master of his own empire.

I started getting to know Rick when we traveled to London to open the Cambridge Lab. Rick was juggling a number of local activities, spending time on long phone calls with Michael Freedman and it seemed to me, he and I were dragging a little, while Nathan continued to draw from some endless energy source.

As Nathan was preparing to move into a CTO role, he asked me if I would work for a particular executive. Uncensored, my response was, "He doesn't read fiction. I can't work for someone who doesn't read fiction."

Nathan's response: "What on earth does that mean? That's crazy. I only read one or two fiction books a year. Why does this matter?"

Me: "It's about suspending disbelief. Especially in research. Fiction requires it. If someone only reads The Wall Street Journal how can they manage research and creative types?"

Soon after that, Nathan let me know that he had been promoted and that I'd be reporting to Rick. "You'll be fine, Linda. Rick was an English major." Nathan continued to find my requirement incredibly weird.

Maybe that's Rick's secret, though, that he was an English major and, along with the technical background, it's an extraordinary combination. He can suspend disbelief. He finds amusement in things others would stress over. He doesn't take himself, or others, too seriously. Lots of wisdom and an adventurous spirit.

Software Development and Engineering Research in Microsoft Research, Redmond

James Larus, Bob Davidson, Yuri Gurevich,
Wolfram Schulte

From its earliest days, Microsoft Research worked on problems in software development. The early efforts, such as BBT, focused on improving the performance of Microsoft's software, but subsequent efforts shifted to the more challenging domain of software quality. Currently, the RiSE (Research in Software Engineering) group is one of the largest in the Redmond lab, and its efforts have changed the way in which software is developed in Microsoft, as well as the agenda of the larger research community. This paper provides a short history of the origins of this work.

The Early Days

When Windows NT 3.1 was released to manufacturing in the summer of 1993, its box requirements were 16MB of RAM. At that time, many machines had 4MB or less, and even high-end machines were running with 8MB. Of course, these high RAM requirements were a concern since 4MB cost over $100 at that time. As the NT 3.1 gold disks were making the trip to Canyon Park, to begin the production process, members from two teams within a young Microsoft Research were hard at work preparing a demonstration for the next day. The MSR Advanced Development Tools (ADT) and OS Research teams had requested a meeting with MS senior management to demonstrate how new tools and techniques from MSR could impact 'real' Microsoft products. The next afternoon, two identical machines, one with the RTM gold bits and a second with an MSR-compiled version of the OS were put through their paces. At the end, the executives agreed that MSR version was running slightly faster than the RTM bits, but for the most part, there were no significant differences. At this point, it was revealed that the MSR version was running with just 12MB of RAM, while the RTM version had 24MB. Using profile-based analysis and optimizations, the MSR teams had cut the working set to less than half of its original size and it *still ran slightly faster*. At the end of that day, MSR was fully engaged and contributing to Windows NT 3.5.

Software being Microsoft's business, it is not surprising that software development tools were the focal point of the earliest groups in Microsoft Research. Bob Davidson's ADT team joined MSR from the product groups during a re-org when their "Lego" project was scheduled to be canceled as "too risky." Later renamed to BBT, "Lego" produced a program executable modification tool that reordered the basic blocks in a program, to enhance virtual memory paging behavior by moving "cold" blocks out of line. Without BBT, Windows 95 and the Office programs would not have run satisfactorily in 4MB of memory, an almost inconceivable idea today, but a key design goal 16 years ago and an early demonstration of MSR's willingness to take a risk and make an impact.

Another early member of MSR was Charles Simonyi, who started the Intentional Programming (IP) project. The rationale for this effort is similar to what used to be called extensible programming languages, and which is now known as domain-specific or embedded languages. Charles believed that programming languages' fixed vocabulary and grammar unduly restricted developers, who should be able to design an embedded language to represent their abstractions. The IP team built a series of programming environments, which, while self-hosted, never fully resolved the semantic and implementation challenges of transforming a sequence of glyphs to executable code. Simonyi left Microsoft in 2002 and started Intentional Software, which continues to pursue this idea.

In 1992, Daniel Weise joined MSR from Stanford to start the Semantic-Based Tools (SBT) group, arguably the first software engineering research group in MSR. SBT had two goals: to develop new program analysis techniques and to build advanced programming tools based on these program analyses to help developers construct correct code. The AST Toolkit added a clean abstraction layer to the Microsoft C compiler's parser, so that it became possible to write simple plug-ins to traverse a program's representation and detect defects. A goal of the AST Toolkit was to open code up to other programs, so developers could easily write, manipulate, and reason about their code. Using the toolkit, the SBT group demonstrated a 'query by example' capability, where a developer identifies a bug, highlights the code in question, and the tool finds all similar constructs. This extensible approach to program analysis was the direct antecedent of Prefast, a similar framework produced by PPRC and still widely used throughout Microsoft for analyzing code and finding defects.

SPT

In 1997, Jim Larus took a sabbatical from the University of Wisconsin to join the BBT group. While at the university, Jim had corresponded with a group in MSR about program executable editing, an area related to his EEL research project. During his year at Microsoft, Jim observed that software developers at Microsoft relied on the same three tools that their brethren at universities and other companies had used for decades: an editor for writing code, a compiler for translating the code into a form computers could execute, and a debugger for examining and controlling a running program to find and understand program errors ("bugs"). The tools were, and still remain, fundamental to software development, but were proving increasingly insufficient as Microsoft struggled to improve the reliability of software that, in the late 1990's,

was becoming part of the general fabric of society. The explosive growth of the internet soon after exposed another type of problem, as malicious individuals intentionally found and exploited software defects to quickly infect large numbers of on-line computers.

In the late 1990's, Microsoft Research started two related efforts to improve software development tools. In March 1999, Amitabh Srivastava started the Programmer Productivity Research Center (PPRC), which built a number of defect-detection tools. Windows NT had bought a small startup called Intrinsia, whose Prefix tool used a variety of techniques and heuristics to find bugs in large code bases. The tool produced detailed reports that were given to developers to understand, prioritize, and act upon. The tool was successful in finding large numbers of simple software bugs and is widely believed to have improved Microsoft's software quality. (Unfortunately, no systematic studies were performed, but Prefix found 1/8 of the bugs in Windows Server 2003.) However, it had several severe shortcomings, including a non-incremental operation requiring several days to analyze a large program and a lack of extensibility that made it difficult to detect new defects. The acquisition brought Jon Pincus and several other people from Intrinsia into PPRC, where they led the efforts to improve this tool. PPRC also performed pioneering work on software fault root-cause analysis, optimizing program testing, and analysis of the dependencies in large systems.

At roughly the same time, Larus started the Software Productivity Tools (SPT) group to develop systematic defect detection tools based on computer science research. The first problem was to continue SBT's research on scalable program analysis, which could efficiently analyze millions of lines of code. Previous work in this field was in the context of compilers, which traditionally analyze and optimize a program in small pieces. SPT's work lead to scalable alias and value flow-analysis algorithms that could analyze complex relationships in millions of lines of code. These techniques were heavily used in later Microsoft tools, particularly as part of finding buffer overruns.

Our other line of work was to find better ways of finding bugs in device drivers, which are complex, low-level parts of an operating system, particularly difficult to write correctly. This research led to a new technique called software model checking, which combined several ideas from hardware verification with ideas from program analysis and led to a new way of finding software defects. The approach was quickly hailed by both the hardware and software community as a fundamentally new insight and has led to a decade of innovative research in applying formal methods to software. More impressively, within five years this research result was the basis for Microsoft's Static Driver Verifier tool, which is widely used and acclaimed by the device driver community, and which has been supported and developed jointly by Windows and MSR.

In the early part of the 21st century, malicious attacks on computers, such as Code Red and BLASTER, exposed the fundamental weakness of the C and C++ languages to buffer over-run attacks and led to the development of tools to find these defects. These tools were built on an extensible framework called Prefast developed by PPRC, but clearly modeled on SBT's earlier AST Toolkit. The security tools required the annotation of function interfaces in millions of lines of .h files, using a language developed by SPT, a process mostly automated using

scalable program analysis techniques from this group. The resulting annotations and tools were incorporated into Microsoft Visual Studio and are available to all Windows developers, both in and outside the company.

Recent development tool research has focused on using theorem proving and SAT-solving techniques to supplant other program analyses. This work builds on the incredible progress the larger research community has made over the past decade in improving the efficiency and scalability of these two fundamental techniques. MSR built and maintains Z3, one of the best theorem provers. It has been incorporated into a variety of internal tools, for defect detection, program testing, fuzz testing, and other uses.

Another key research direction of SPT was to study the developers and testers who produced the software, not just the artifacts that they produced. The Human Interactions of Programming (HIP) group was one of the first efforts to systematically study the software development process at Microsoft and the interactions among the people who participate in this endeavor. Subsequently, the Empirical Software Engineering group applied statistical analysis and modeling techniques to the vast amount of data collected as part of this process, to help understand which components of a system were least reliable and why. The tools and techniques pioneered by this group fundamentally changed software development at Microsoft.

Another recent area of work is tools for finding defects in concurrent and parallel programs. MSR's CHESS and various derivatives pioneered a new approach to systematically testing parallel programs and has been used by several groups to find subtle, low-level bugs in shipping software. MSR has also built automated testing tools to find program defects by combining static program analysis with guided execution, an approach that has been extremely successful in finding security flaws in programs and has helped keep Microsoft ahead in the race with attackers using simpler, random techniques of program fuzzing.

FSE

Yuri Gurevich joined MSR in 1998 and started the Foundations of Software Engineering (FSE) group. Wolfram Schulte joined FSE in 1999 and took over leadership of FSE in 2003. For the first five years, almost all efforts of the FSE group concentrated on one project: executable specifications and model-based testing of protocols based on the theory of abstract state machines. The research challenges were how to express and how to compose models, how to deal with concurrent and/or nondeterministic agents, and how to prune and guide the exploration. Wide-scale experiments support the claim that model-based testing is more cost effective than manual testing. In 2007, Spec Explorer was transferred from Microsoft Research to the Windows organization, to help with the Technical Document Testing Program of Windows, mandated by the US Justice Department as part of Microsoft's compliance with a 2002 antitrust settlement. Since then Spec Explorer has been used to generate conformance tests for 200+ protocols, requiring more than 200 man years to develop.

In 2000, Schulte and Meijer investigated how to build three-tier applications consisting of a GUI, business logic, and a database, written in HTML, C# or Java, and SQL, respectively. The

state of the art at the time was to use strings to glue the different tiers together with obvious performance (marshaling is expensive) and security issues (like SQL injection attacks). Their insight was that it should be possible to extend an OO language to incorporate features from the XML and SQL scenarios (syntax and types) to produce a uniform way to develop those systems without the marshaling overhead but with guaranteed security. The X#/Xen/Cω projects extended C#'s nominal type system with structural types that captured SQL and most of XML and introduced queries over arbitrary object graphs. In 2004, the Cω proposal was merged with Anders Hejlsberg's C# sequence operator project. The joint project became known as Language Integrated Queries (Linq), which shipped in Visual Studio 2008 and is one of the key research contributions for C#. Linq has meanwhile transformed the query ecosystem: pLinq executes in parallel on multi-cores, Dryad Linq executes distributed systems, and Rx evaluates standing queries.

In 2003, the Spec# project started looking at the question: w*hat is the meaning of object invariants in the presence of inheritance, call backs, aliases and multi-threading*? Spec# is the traditional verifier for C# using verification condition generation and an automatic first-order logic prover. It also makes several novel contributions, ranging from delineating when invariants have to hold, to an ownership type system that describes what invariants can depend on and what can be changed by method calls. In the context of Spec#, many other verification challenges have been addressed, such as non-null types, purity, and model fields. Spec# was also built to demonstrate that a practical verification system can be used at design- and run-time. Unfortunately Spec# was not adopted for .NET, since .NET is language-agnostic. But, in 2007 we introduced Code Contracts for .NET, a language agnostic library for writing contracts. The Code Contracts library has meanwhile been adopted by product groups and ships as part of Visual Studio 2010. .NET is thus the first large commercial platform that exposes contracts. VS 2012 will expose them in MSDN. Design time integration (i.e., the static checking of Code Contracts) is planned as part of the next version of Visual Studio.

In 2005, our main programming model for multi-core machines was still threading. Unfortunately, threading is difficult to use—when parallelizing a sequential application you have to not only identify the potential parallelism, you also have to divide the work, allocate it statically or dynamically to threads, maybe introduce mutual exclusion, and finally rewrite the algorithm according to the new control structure. Simple sequential algorithm might become twice as long and several times as complex. FSE built the Task Parallel Library (TPL), which uses higher-order functions and generics to build new control abstractions for parallel programs. TPL shipped officially as part of "Parallel Extensions to .NET" in Visual Studio 2010. TPL has now become the centerpiece of the Microsoft parallel story: parallel LINQ is implemented on top of TPL and the new async extensions in C# version 5 use the TPL in their implementation. Also Microsoft's C++ parallel patterns library is very closely based on the TPL API and uses the same techniques.

Conclusion

MSR's research on software development and engineering is central to improving Microsoft's

core competency of software development and improving the quality of its products. MSR has a long history of building tools that have both improved development within Microsoft and provided more advanced programming tools to aid our customers in writing their applications. The breadth and diversity of these efforts mirrors the complexity of software development and has evolved to respond to new challenges, such as memory pressure, security, Web development, parallelism, and the new areas of cloud and mobile development.

Rick Rashid, Microsoft Research, and Academia

Ed Lazowska

Bill & Melinda Gates Chair in Computer Science & Engineering, University of Washington

One of the great privileges of my professional career has been my association with Microsoft Research since its inception.

Nathan Myhrvold asked me to come over one day in late 1990 or early 1991—at a time when Microsoft's annual revenues were only a bit more than $1 billion and its headcount was only a bit over 5,000. He described a typically Nathanesque idea (Nathan not being known for small thinking): he was going to recruit a leader who would create a world-class computing research organization. He invited me to serve in an advisory capacity, along with Gordon Bell and a few others. I jumped at the chance.

Within a year, Nathan had recruited Rick Rashid. This effort included recruiting Rick's kids, by sending them box after box of paraphernalia from the Woodland Park Zoo. (When Nathan decides that you're his first choice, you'd better be prepared for a full court press.) And the rest, as they say, is history: Microsoft Research, now in its 20th year, is indisputably the world's leading computing research organization.

My focus here is not on that history, though, but rather on the relationship between Microsoft Research and academia.

Of great importance, Microsoft Research publishes essentially all of its work in the open literature. Microsoft's annual investment in Microsoft Research is in the same ballpark as the National Science Foundation's annual investment in the CISE Directorate, which provides more than 80% of the support for academic computing research in this country. In other words, Microsoft is spending about as much as the National Science Foundation to drive the field forward, and very little of this is for proprietary work. It was not a foregone conclusion that Microsoft Research would be an integral part of the national and international computing research community—it was a principle that Rick established.

So, yes, Microsoft Research contributes hugely to the field. Microsoft Research also contributes hugely to academia. I could describe the Microsoft Research PhD Fellowship program, which each year provides generous 2-year fellowships to roughly 10 top graduate students. Or the Microsoft Research Graduate Women's Scholarship program, which each year provides 1-year fellowships to roughly 10 top woman graduate students. Or the Microsoft Research Faculty Fellowship Program, which each year recognizes eight of the most promising young faculty members from around the world and provides them with very substantial research support. I could describe research tools such as the Kinect SDK, or the extensive research grants program, or the undergraduate and graduate student internship programs, or the sabbatical visitor program, or the financial support provided to literally hundreds of technical conferences each year, or the long-standing support of gender diversity efforts such as the National Center for Women & Information Technology. I could talk about the collaborations—in the case of the University of Washington, 25 Microsoft Research staff members who serve as Affiliate Professors, teaching classes and advising students and creating wonderful innovations such as Photosynth.

But I'm going to focus, instead, on Rick's integrity in dealing with universities. I'm going to do this through two examples.

The first example concerns the University of Washington. Rick and I had a conversation in 1991 about the future of the University of Washington's computer science program. We were a lot more fragile back then than we are today. Rick understood the importance of a strong academic program to the Pacific Northwest. Among other things, he committed that in building Microsoft Research, he would not recruit faculty from UW. Rick has adhered scrupulously to this pledge, even as other companies have been less principled. In 1999, Microsoft acquired UW professor David Salesin's startup company Numinous Technologies, and David along with it—but Rick granted 50% of David's time back to UW. A decade later, when an unnamed company offered one of our faculty members a lucrative deal to move to that company, Rick offered him an even more lucrative deal to remain at the University of Washington. Microsoft in general, and Microsoft Research in particular, are the University of Washington's greatest strategic assets. Together, we're working to make the Pacific Northwest a global center for computing research. Rick's commitment to preserving and enhancing the strength of the University of Washington has been essential.

This same degree of "enlightened integrity" has played out around the world, which motivates my second example. When Microsoft Research Asia was established in 1998, the Chinese university system was still in recovery. There were three strategies that Microsoft could have adopted: build up MSR-A by recruiting the few Microsoft-caliber researchers who were then working in Chinese universities; staff MSR-A with returning expatriates while waiting for the Chinese university system to recover on its own; or put programs into place to dramatically strengthen the Chinese university system. Not surprisingly, given Rick's leadership, MSR-A chose the third option. Fresh Chinese PhDs (as well as some from Hong Kong and other Asian countries) were hired as "Associate Researchers"—essentially, postdocs—mentored for several years until they had the capabilities of fresh PhDs from top U.S. schools, and then

returned to Chinese universities as faculty members. This has contributed enormously to the dramatic strengthening of academic computer science programs in China.

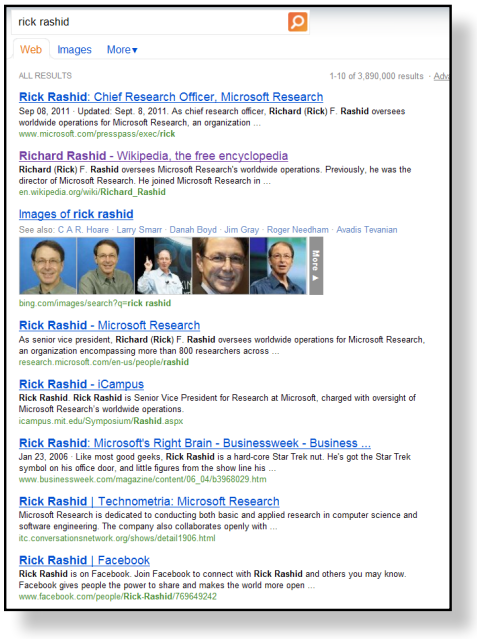
Rick Rashid has his head screwed on right. The beneficiaries include not only Microsoft, but also the University of Washington, and the national and global computing research communities. Thanks, Rick, for your vision and for your integrity.

Rick Rashid's Impact on Microsoft Product R&D

Xuedong Huang

Talk at Rick Rashid's Festschrift
March 9th 2012

Rick Rashid was my first manager when I joined Microsoft Research in 1993 to start our nascent speech technology team. During my 19 years at Microsoft, I have been in and out of MSR several times and restarted my professional career with each move (aka first helping to bring speech technology to mainstream, incubating a few startups including Response Point, and now serving as the chief architect for Microsoft Advertising R&D). One thing that never changed is Rick Rashid's lasting impact on me personally, on so many friends around me, and on so many great tech transfer stories from MSR.



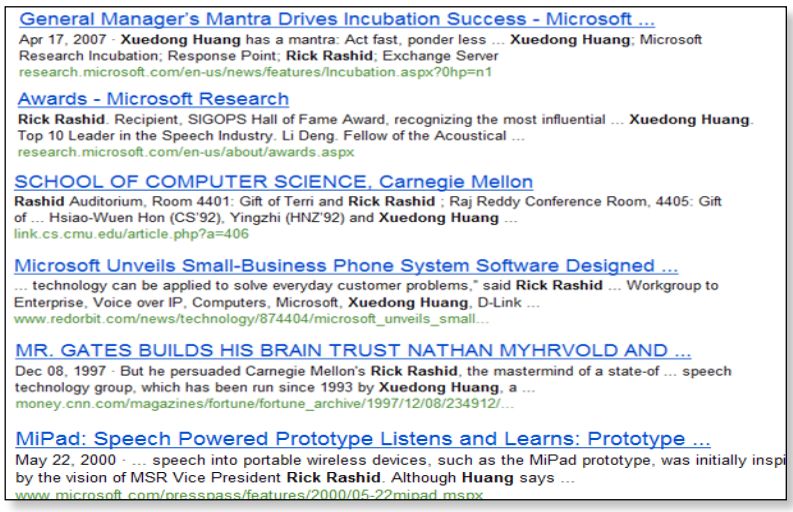
As I'm working on search & advertising, I can't resist searching on Bing for Rick's product impact, shown here. I don't want to repeat his very impressive contributions to the company's product offerings. I will share a few personal stories on Rick's approach and influence on tech transfer and our product R&D.

Building the team: When I started in 1993, I worked with Rick to build a world-class speech technology team. Rick created a framework that blends university and industrial research culture. Rick helped me recruit most of the senior folks I hired into Microsoft. Interestingly, most of them are still at Microsoft today. By start date: Fil Alleva, Mei-Yuh Hwang, Alex

Acero, Li Jiang, Hsiao-Wuen Hon, and Ye-Yi Wang. Rick ensured we had the best people we could bring to Microsoft. The way he inspired people around him and the real opportunities he provided to people around him worked very well. I'm so pleased that most of my early hires are still playing very important roles in Microsoft today, thanks to MSR's culture.

Passion and vision: Rick has his magic touch. Making speech mainstream was the vision Rick helped us create to move the state-of-the-art speech technology forward. That vision and passion helped us introduce the first Speech API to the public on Windows 95 (yes, we shipped 4 versions of SAPI from MSR since Windows 95). All of today's speech solutions in Windows, Office, Phones, Server, Bing, and Xbox can be traced to what we delivered in MSR.

Tech transfer as a full-contact sport: With MSR's pioneering work on speech R&D for many years, Microsoft decided to establish a sizable speech product group. I had the privilege to lead the newly formed speech product team as General Manager and I left MSR in 2000. Thanks to Dan Ling and Rick's sponsorship, the whole speech research group came with me and joined the new product team for several years. The research team played a key role to help Microsoft speech get up to speed. This was a great example of practicing Rick's philosophy that tech transfer should be a full-contact sport!



http://www.microsoft.com/presspass/features/2000/05-22mipad.mspx

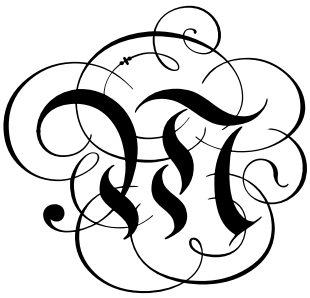
Long-term focus: Rick always encouraged all of us to take more risks working on research projects that could alternate the future. One thing that stood out was MiPad that we incubated in MSR. We partnered with HTC and created

a compelling prototype that Bill Gates demonstrated in 2001's CES. I searched on Bing again and enjoyed reading what we discussed in 2000. I told the press that "We're selling it as a vision for Microsoft and the industry". It is unclear if Steve Jobs studied MiPad or not, but we do have rights to "accuse" Apple of taking M out from MiPad for its own flagship product. Keep in mind that Apple is blocking a Chinese company from using ePad. ☺

Huang explains that incorporating speech into portable wireless devices, such as the MiPad prototype, was initially inspired by the vision of MSR Vice President Rick Rashid. Although Huang says it might be awhile before MiPad and similar Microsoft products become available, he sees benefits beyond commercial release. "I would love to see MiPad technology turned into a product," Huang says. "But we're not just selling it as a device. We're selling it as a vision for Microsoft and the industry."

Leadership: Thanks to Rick's leadership, MSR has had a very deep impact on so many product groups via both technology and people transfer. From beating Google's crown jewel with the Speller used in Bing to our attempt to redefine a web interaction metaphor (Project O: HyperText → HyperTEC), OSD teams enjoy working with so many world-class researchers to solve hard computer sciences challenges. After I took the role of chief architect for Microsoft Advertising R&D, a few key initiatives I'm currently working on have also benefited from MSR researchers' deep involvement. MSR researchers are not only great at solving the hard computer sciences challenges but also great at simplifying business and monetization complexities.

THANK YOU AGAIN, RICK, AND HAPPY FESTSCHRIFT!



anaging Microsoft Research

On Software Principles and Research Management

Roy Levin

Managing Director and Distinguished Engineer,
Microsoft Research Silicon Valley

arge software systems are complicated, as are large organizations. Complexity naturally arises as both grow, but in complexity resides the seed of dysfunction. Techniques for dealing with complexity, or for preventing it from arising, almost invariably involve subdivision of the whole in a principled way: divide and conquer, separating mechanism from policy, distinguishing the norm from the exception, etc.

In software systems, a key technique of this kind is *modularity*, which provides a disciplined way of organizing software for long-term manageability and development. *Information hiding* and *well-specified interfaces* are two closely allied principles. In a software system, the wise architect/developer packages the functionality into *modules* that interact through clearly specified channels (interfaces) and encapsulate implementation details. Modules relate to each other through *abstractions* (represented in interfaces) that capture what modules may assume about each other (*dependencies*). These ideas of software structuring go back to the works of Parnas, Dijkstra, Hoare, Wirth, and others in the 1960s and 1970s, and are rooted in the fundamental desire to manage the complexity that increases rapidly as software grows.

Management of complexity in organizations, especially research organizations, is essential for growth, too. Why research organizations especially? Research depends for its success on the ability and freedom of creative people to do what they do best: generate and explore ideas. But organizational growth requires structure to avoid chaos, and structure tends to limit creativity. A wise research manager will impose the minimum of structure and exploit the principles that control complexity in large software systems to keep the organization functioning as it expands. Modularity and related principles are especially relevant, as the next few sections will elaborate.

The management of Microsoft Research (MSR), under Rick Rashid's leadership from its inception[1], has applied software principles effectively, if perhaps not always consciously.

In this short paper, I point out some instances in which the principles of modular software structure listed above, as well as some of their corollaries and cousins, have been applied to enable MSR to operate and grow successfully over the past twenty-plus years.

While I think the parallels between software structure and research management are real, I recognize that the analogy can be pushed too far. Software systems and human systems are different animals, and though they have areas of significant and interesting overlap, they also are obviously distinct in many ways as well.

Support Functions

Microsoft Research is a geographically distributed organization with a shared, common mission: to advance the state of the art in computing and to transfer those advances to Microsoft's products and services. It is customary in many organizations, and MSR is no exception, to compartmentalize certain complex and relatively well-delineated functions—Finance, Legal, HR, general administration—and to provide a simple interface for the rest of the organization to access those functions. In MSR each of these functions is distributed geographically, but that implementation detail is largely invisible to the users of those functions. This works well because each function usually has some (geographically) localized aspects (accounting rules, patent law, employment regulations, etc.) layered on shared, common rules and processes. Via a well-defined interface, the delivery of the function is organized internally to hide this layering, which can be complex, from the user.

This approach also echoes a principle of *internationalization* in software: to factor out and compartmentalize (through interfaces and related structuring techniques) the geographically unique aspects of a software component, leaving a broadly applicable common and coherent core. Additionally, geographic distribution reduces latency (from time zones) and performance bottlenecks at the core by enabling some functions to be performed entirely locally.

Product group interaction and technology transfer

Microsoft Research has about 2% of the people in Microsoft. To make the mission of delivering innovation to the product organizations feasible, MSR must provide usable interfaces that encapsulate the complexity. The MSR Program Management (PM) team is a major component of that interface, serving as a two-way "connection set-up" mechanism. Researchers seeking an outlet for their innovations work with (generally local) members of the PM team. No single member of the PM team can possibly track opportunities across the entire company, so the implementation behind the local interface is a mesh through which, utilizing at most two PMs, a researcher can be connected to a potential technology transfer opportunity in a product group. Similarly, for a product group team seeking researcher expertise in a particular area, the MSR PM team provides a single contact person (interface), who internally uses the same mesh in the opposite direction to reach one or more relevant researchers. This implementation also accommodates geographic distribution of talent transparently.

It should be noted that this connection mechanism is typically just that: a way of finding the

right connection. Once the connection is made, communication can occur directly between the endpoints without involving the MSR PM team. This bypassing of layers to obtain efficiency recalls a common analogous software structure. As with that structure, the efficiency is a trade-off, since by bypassing the management mechanism the status of the communication becomes invisible to the management, making error detection (e.g., lost connection) and recovery more challenging.

External relations

As part of its mission to advance the state of the art in computing, Microsoft Research supports and conducts a number of activities involving academic and professional institutions around the world. Once again, the principle of well-defined interfaces applies; for this purpose, it is the Microsoft Research Connections (MRC) group within MSR. Like many of the internal support functions noted earlier, MRC is geographically distributed to enable it to address efficiently and naturally local needs and interactions, but with a common core that coordinates strategy and significant funding activity. For many of its activities, MRC is to academic and other research institutions what the MSR PM team is to internal product groups: a simple interface to a comprehensive "connection set-up" mechanism. And, like the PM Team, MRC provides relationship management once the connection is established, which is more commonly needed for external relationships than internal ones. For other activities, it provides functionality (e.g., research grants, fellowships) directly rather than establishing connections with others.

Research collaboration

MSR's organizational structure for research collaboration departs sharply from what it uses for the functions previously discussed. It's tempting to characterize it as "no structure", since the management is a strong proponent of collaboration and is willing to support practically anything that works. The operative word is "practically." Just as within a software component there is benefit in observing the discipline of modularity and interfaces, so within the research organization there are similar benefits. As a result, most research projects are relatively small and focused, with little need for formal management structure within the project. When occasional large projects arise, a leader is generally designated, and there may also be a researcher who acts as a *de facto* architect, designing the modular structure and interfaces.

Generally, the management applies a light touch in shaping research activity, since to do otherwise would undercut the benefit of hiring creative people. But some management attention to research collaboration is necessary if it is to proceed productively. By analogy, consider what happens when two developers work on the same software component without communicating. There is a significant chance that they will duplicate work to some extent, each writing something that they find necessary but which could be made common if they were aware of each other's activities and efforts. So it is with research projects. The job of the management is to ensure communication across portions (especially geographically separated portions) of MSR so that overlapping or duplicative research work is noticed and a deliberate decision is made to merge projects or keep them separate. The management accomplishes

this task with activities analogous to design reviews, periodically sharing information about the work underway (and, in some cases, projected) with the goal of identifying collaborative opportunities and preventing unnecessary duplication. It's also tempting to draw an analogy with principles of *multi-threading programming*: keep state separate when possible, establish clear synchronization points in which information that should be shared is exchanged in robust ways, and keep such synchronization points to a minimum consistent with overall correct and efficient execution. And perhaps one more: beware of deadlocks!

Change

All software systems undergo change. Small change is often continuous and the people that develop a system have well-worked-out and familiar ways to accommodate it, so much so that it is nearly second-nature. But larger-scale change, such as redesign of interfaces and/or refactoring of implementations, is more disruptive, requiring deliberate focus on the act and mechanism of change, often to the exclusion of other activity for a time.

So it is in organizations. In MSR small changes, such as the hiring of interns, occur very frequently and are smoothly accommodated. But larger changes, such as reorganization of the management tree, are potentially disruptive and therefore carefully managed. This is especially important in MSR because, unlike product organizations, research has and depends on stable management structures over extended periods, which enable researchers to take long-term risks confident of continuing management support. Thus change, when it comes, must be carefully handled.

The MSR management approaches organizational change much as a development group approaches a substantial software change. There is a planning period in which only a few key people are involved, who design the overall change and think through the implications. The change might involve reassigning people or reshaping the reporting hierarchy, or it might be a change in organizational policies or procedures that affect a large fraction of MSR. Once the shape of the change is determined, additional people are enlisted to work through the details of the implementation; these are the people who will be most involved in effecting the change itself, not the larger group who will eventually need to respond to the consequences of the change. Throughout these stages, confidentiality is important, since rumors can undermine or even wreck what the change is intended to accomplish. This, incidentally, is where the analogy with software systems breaks down, since code doesn't gossip. That difference also influences the next stage of the process, which is prompt communication and implementation. The longer one waits to communicate a worked-out plan, the greater the possibility for inaccurate information to get around. Also, the longer one takes to implement a change in which individuals take on new responsibilities, the greater the loss of organizational momentum while people occupy "lame duck" roles.

Dealing with the unexpected

By their very nature, abstractions omit aspects of reality—they simplify it. Occasionally (and perhaps inevitably), reality intrudes to the extent that an abstraction fails. In a robust software

system, planning for the unexpected is essential, so that the user of the system can rely on it for dependable, or at least understandable, operation. Planning for the unexpected means writing *exception-handling* code that will be used only in exceptional situations. Developing confidence that exception-hunting code will work properly when it is needed is a major aspect of testing methodology.

In organizations, the problem of dealing with the unexpected is both easier and harder than in a software system. On the one hand, the response to an exceptional event may (and often does) involve human judgment which, unlike a software exception handler, need not be detailed in advance. On the other hand, noticing that an exceptional event has occurred is usually easier in software, partly because the number of things that must be checked for is smaller than in human affairs. This means that, in the latter, an exception often goes undetected longer, causing other abstraction failures before it is finally noticed and making the subsequent recovery that much more difficult.

In MSR, some aspects of the unexpected go with the territory, since MSR is a research organization and research necessarily entails exploring the unknown. Researchers are trained for this, and unexpected technical problems are thus not exceptional. (One might say, when it comes to technical matters, research involves the "expected unexpected.") However, non-technical surprises arise too, and it is important to have simple and robust exception handling mechanisms for them. This starts with detection, which is a globally-shared responsibility of management and individual contributors. It depends crucially on managers knowing what the researchers are doing at a sufficient level of detail to detect the potential for non-technical surprises. For example, if a researcher is planning to give a talk in an unfamiliar venue, he/she may be asked to sign an unfamiliar document—perhaps a non-disclosure agreement or a video release form. The researcher will recognize this as an exceptional condition when it occurs, but may find it challenging to address at that moment. Communication with management in advance—say, "I'm giving a talk to the NSA next week"—will result in smooth handling of the issue before a real-time crisis occurs.

In general, the exception-handling action that all researchers know (and nearly always invoke when necessary) is "Ask your manager." This transfers the problem to someone who is trained in a wider variety of policy and procedural matters than researchers know or want to know. One can regard this as an instance of the modularity principle as well, since it encapsulates the details of handling all sorts of exceptions in one place—the manager "abstraction." That exception handler isn't solely the manager but is actually a complex network of support functions that the invoker—the researcher—doesn't need to understand. The implementation of that exception-handling function involves a mixture of prior manager training, subject matter experts in the support functions on whom the manager can draw, corporate policies, and when all else fails, common sense.

While this non-technical exception handling occurs in all sorts of organizations, MSR takes a somewhat unusual—one might say counter-cultural—step to help make exception-handling robust. In most of Microsoft and in many corporations, the trend is to reduce administrative support by deploying online tools that employees use directly for a variety of functions:

expense reporting, travel, HR support, and so on. When done well, these tools can provide great efficiency for employees while reducing labor costs for the company, but sometimes the tools are difficult to use, inadequately supported, or are built on assumptions that don't hold in MSR. In these cases, MSR either deploys its own tools or, more often, invests in additional administrative help in order to lighten the load on researchers. This is an appropriate tradeoff, since it reduces the distraction of researchers from their creative responsibilities, thereby increasing their value to the company. As a result, exception-handling in many of the administrative activities that employees have to do is performed by people within MSR who are familiar with both the individuals and their needs. The result is robust for both the researcher and the organization.

External validation

How does a research organization measure the quality of its work? In academia, there is a standard answer: external peer review. In the academic setting, peer review applies papers submitted for publication and proposals for research funding, and is periodically invoked to evaluate academic departments in ways not covered by the previous two. No academic research institution is taken seriously by the professional community unless its work is regularly reviewed by qualified members of that community.

In some corporate research settings, peer review is not used to evaluate research quality. In MSR, however, peer review is taken seriously and used extensively. Research is published in the same venues that academics use, and visiting committees of academic luminaries (called Technical Advisory Boards) regularly assess MSR's research program and provide evaluations to the management. But peer review doesn't stop there. The annual performance review of researchers is routinely informed by constructive criticism from research colleagues, both internal and external, with the goal of increasing quality of future research and therefore its impact. One can draw an analogy to the improvements that arise in software development through methodologies like "buddy programming," "egoless programming," agile development, and open source. The details differ, of course, but the central idea that shining external light on the development process produces a stronger result applies to research work as well. The MSR management creates opportunities for researchers to expose their work to their colleagues across the organization, such as the annual TechFest that resembles a large-scale science fair. Such opportunities represent an informal but valuable form of peer review, one that occasionally even leads to new research collaborations.

Summary

Many of the principles and techniques of software development have analogs in the management of a research organization. Perhaps this should not surprise us, as the conduct of research and the production of software are both creative endeavors. Even if we aren't surprised, the extent of the analogy, as suggested by this paper, should perhaps motivate us to think, as we strive to improve and grow the impact of research, what other principles from software creation may have useful analogies for management.

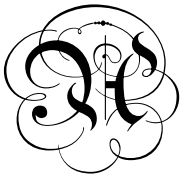
Endnotes

1 Rick has had a remarkable and distinguished career as a professor, researcher, software creator, and research manager and corporate executive. During substantial portions of that career he has "gotten his hands dirty"—doing the hands-on work of building software—and during other substantial (and sometimes contemporaneous) portions he has guided the activities of others in conducting research and software creation. I suspect that, in doing so, he has consciously or unconsciously applied the same or similar principles to these endeavors. This paper explores the applicability of software development principles to research management, and while it doesn't make specific connection to things that Rick has done, I trust that many who read this paper and have worked with Rick will recognize them.

Rick Rashid & Microsoft Research Asia

Hsiao-Wuen Hon, Chuanxiong Guo, Jian Sun, Xin Tong,
Lintao Zhang, Yongguang Zhang, Feng Zhao, Lidong Zhou

"Half a world away from the calm beauty of Seattle and Puget Sound, there's a lab where software dreams come true. At Microsoft Research Asia, the drive to succeed is as intense as the traffic that roars by the front door in unbridled, chaotic fury. If Microsoft's other facilities around the globe seem idyllic, this one, in Beijing, China, is pure street. Nearby high-rises compete with smokestacks for skyline supremacy. Run-down buildings sit next to bustling consumer electronics markets and the Beijing Satellite Manufacturing Factory, where China conducts its spaceflight research. Microsoft's mantra: work hard to get in the door; work harder to survive; then work even harder because the real work-that of an information technology world leader-is just beginning….." —MIT Technology Review, June 2004 [1]



defining moment for Microsoft Research Asia (MSRA) was when it was featured on the cover of the MIT Technology Review as the "World's Hottest Computer Lab" in June 2004. Dr. Rick Rashid had foreseen this incredible opportunity six years earlier when, in one of his career's many pioneering efforts, he started Microsoft's basic research lab in the Asia-Pacific region. Since its founding in November 1998, Microsoft Research Asia has attracted more than 200 top-caliber researchers and scientists from all over the world, supplemented by a Post-Doc research center and over 2,500 visiting researchers and students. For more than a decade, MSRA researchers and engineers have been pushing the boundaries of innovation and contributing significantly to Microsoft and the broader community.

Today starting a research lab in China is quite obvious and common. However in 1998, it took extraordinary pioneers like Rick to launch this ground breaking effort. MSRA was the first research lab established by a multinational company in China. It was also very unusual

that Microsoft established a research lab before it had product development efforts in China. MSRA, under Rick's leadership, actually helped Microsoft incubate several successful product development organizations in China. In fact, until very recently, Rick was the Microsoft executive sponsor for Microsoft's entire research and development efforts in China.

In addition to leading MSRA, Rick's technical influence has played a vital role in shaping and fostering its technology portfolio. Here, I would like to highlight Rick's technical influence on several key areas at MSRA as much of our best research can be traced back to his earlier pioneering work. It has been a constant source of pride for us.

Computer Vision and Graphics Research at MSRA

MSRA first gained its reputation as a world-class research lab in the area of multimedia technology, particularly computer graphics and vision. Rick's PhD thesis [2] at the University

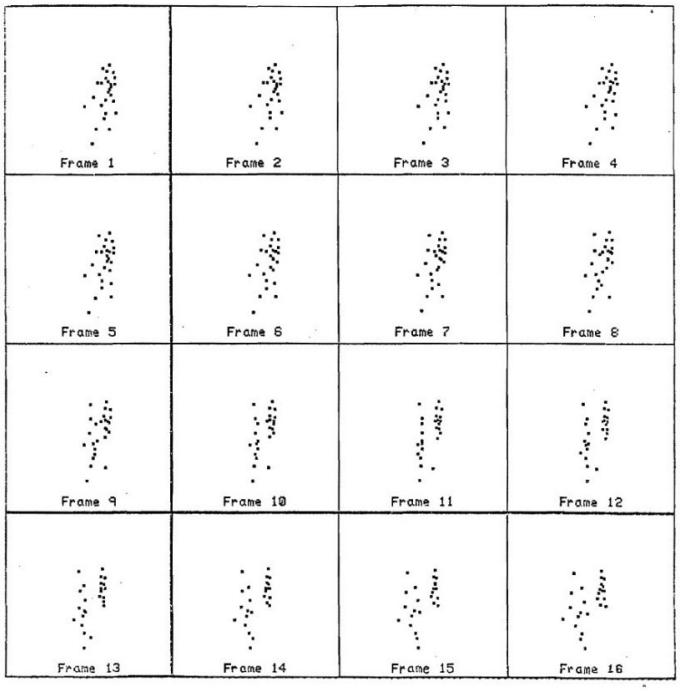


FIGURE 1: "Two men walking" moving light display in Rick's PhD thesis [2].

of Rochester was a pioneering study of the perception of human motion with moving light displays in computer vision. Rick's work illustrated that it is possible to reconstruct the human motion of a non-rigid human body from very limited observations (unlabeled 2D moving display points) and thus presents many exciting new research challenges and opportunities.

On the one hand, this work stimulated more study of the intrinsic properties of human motion

and thus enabled much new research in motion synthesis, tracking, and understanding. On the other hand, it also inspired systems and applications for capturing and reconstructing human motion, which eventually resulted in the wide availability of motion-capturing systems in the movie and entertainment industry.

Motion Synthesis & Tracking

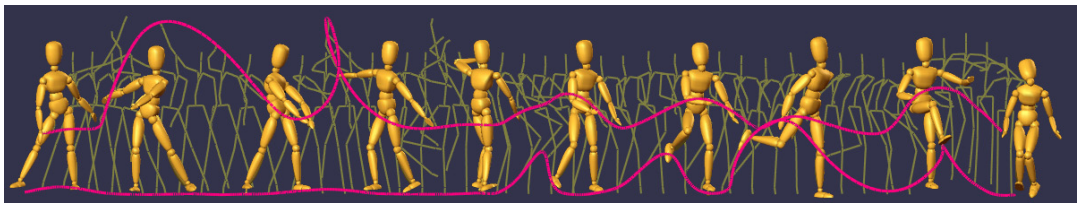


FIGURE 2: This 320-frame sequence of dance motion is choreographed from motion texture from motion captured dance data in [3]. Four motion textons are generated from the motion texture and then used to synthesize all the frames in this sequence.

Rick's thesis work directly tackled the most fundamental challenge in computer vision—interpreting the meaning of visual imagery. To attack this difficult problem, Rick developed a series of algorithms for tracking points between frames, grouping points to separate individual subjects, and dividing each subject into parts. This exciting and challenging research also attracted a great deal of attention from graphics and vision researchers inside Microsoft

FIGURE 3: Non-rigid 3D face tracking by a set of feature points in [4].

Research Asia. In 2002, Harry SHUM, et al. [3] found that the dynamics of human motion within short time clips can be modeled by a linear dynamic system, while longer motion is always composed of repeated components. Based on this observation, they proposed a "motion texture" technique for synthesizing the realistic motion sequence of the human body from captured motion data, such as dancing. This technology has been successfully incorporated into Bing video search.

Thirty years after Rick, we faced a related compute vision problem: interpreting a set of facial points to understand human facial motion. Not surprisingly, we developed a successful solution by following essentially the same methodology proposed by Rick. In our system, we first detect three types of feature points (semantic, silhouette, and salient) and track these points

between frames. We group them using both spatial and temporal information to separate multiple faces. Then, the points of each face are fit into a non-rigid 3D deformable model so that we are able to finally interpret the individual facial parts (e.g., eye, mouth). The work [4] was published in ECCV 2008 and the technology was transferred to the Avatar Kinect [5] and Microsoft LifeCam [6] product lines for video effects and exposure control.

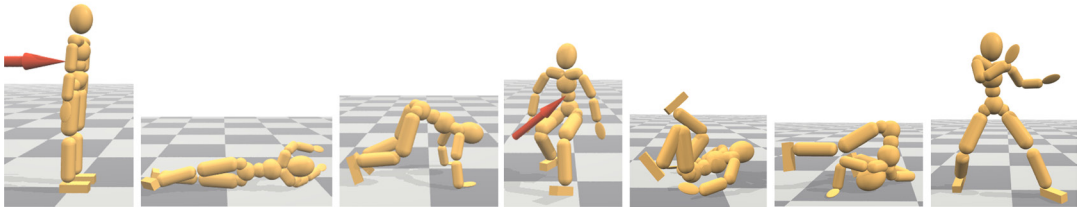


FIGURE 4: An example interaction with the virtual character in [7]. The character gets pushed and falls, rolls sideways, gets up, gets pushed again,rolls backwards, and gets up again.

In 2010, Kangkang YIN et al. [7] combined the motion data captured from the real world with the intrinsic physical rules of human motion to model the complex interaction between humans and diverse environments.

Their first-of its-kind system allows people for the first time to automatically adapt captured motion to different environments and it generates realistic motion results.

Motion Capturing & Reconstructing

FIGURE 5: High-fidelity facial expression with realistic dynamic wrinkles and fine-scale facial details is captured in [8]

Most recently, Xin TONG et al. [8] worked with Jinxiang CHAI from Texas A&M University to develop a new technique for capturing high-fidelity 3D facial expressions. A key observation was that despite the rich temporal and spatial variations distributed over the human face, the human facial expression is actually highly correlated and thus can be well represented by a set of representative face expressions. By leveraging the 3D scan and motion capture data, the system captures high-fidelity facial expression with both high temporal and high spatial resolution.

Systems and Networking Research at MSRA

Rick moved to systems after becoming a professor at Carnegie Mellon University. He led much seminal work in systems at both Carnegie Mellon and Microsoft. Microsoft is primarily a systems company as a provider of computing infrastructure and platforms for consumers and enterprise. Of course, MSRA has deep aspirations to develop a strong presence in systems and networking. However, this task has been a challenge due to the relative scarcity of top Chinese researchers trained in systems and networking area. I have always been of the opinion that a research lab under Rick would never reach world-class status until it built a world-class systems and networking team and portfolio.

Fortunately, under Rick's leadership, we have built core competencies over the past 10 years in distributed systems, storage, tools, and wireless and datacenter networking. We have advanced the state-of-the-art through the impact of our publications at the top systems and networking conferences, and have been honored with three best-paper awards and five best-demo awards. We have provided thought leadership and delivered significant innovations to Microsoft business groups, supporting the company's strategic bet on systems/networking platforms and infrastructure. Finally, MSRA has been working with academic institutions to build a systems and networking community in the Asia-Pacific region. It is a great feeling to know that we've finally made it!

Sora Empowers a Wireless Revolution with the Magic of Software

Software Radio (or Software-Defined Radio, SDR) is an engineering vision for our time. It envisions a future world where different special-purpose wireless equipment is replaced by a single piece of hardware running different software programs. Realizing this dream, however, requires breakthroughs in system architecture design and software technology research. One key issue is radio software's inability to run on general-purpose processors and support the speed requirements of current wireless protocols.

Recently, MSRA advanced this technology with a new system design called Sora [9]. It allows researchers and engineers to build high-speed wireless systems (such as next generation WiFi and 4G) on commodity PCs, instead of relying on time-consuming, expensive, and specialized hardware and chips. It addresses the problem of software performance with techniques that leverages a variety of features common in today's multicore architecture. For example, it replaces computation with in-cache table lookups, exploits vector capabilities, and dedicates certain CPU cores exclusively to real-time signal processing. As a result, it can achieve performance that is orders of magnitude faster than prior art, and it is the first software platform to fully implement IEEE 802.11 b/g on standard PCs. Furthermore, Sora presents a new programming model that combines vector programming, modular design, multi-core programming, and real-time control. Complex wireless communication systems can now be built effectively using C/C++ programming in user mode. This has accelerated research projects in wireless technologies such as MAC, PHY, and MIMO.

Sora gained immediate recognition in the wireless community, winning a number of academic

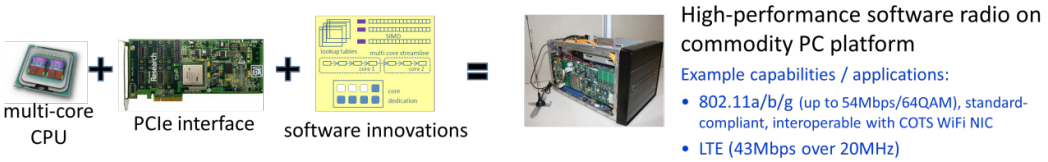


FIGURE 6: The functional architecture of Sora (Software-Defined Radio) [9].

awards, including NSDI'09 Best Paper & Best Demo [9], and SIGCOMM'10 Best Demo [10]. The Communications of the ACM (CACM), the monthly journal of the Association for Computing Machinery (ACM), highlighted Sora in its January 2011 issue [11] calling it "one of the most significant wireless papers in the past few years." Aiming to change how wireless technology is researched and developed, Microsoft Research has released Sora to the academic research community since mid-2010. Currently more than 200 Sora Academic Toolkits have been used by 40 universities and academic institutes around the world.

Data Center Networking Research

The IT industry is experiencing a paradigm shift with the move towards cloud computing, in which a huge number of servers are hosted in large data centers. A data center network (DCN) is indispensable infrastructure for data center applications and services, all of which are bandwidth hungry, latency sensitive, or both. MSRA has been carrying out various DCN research projects, including those focused on DCN architectures, protocols, technologies

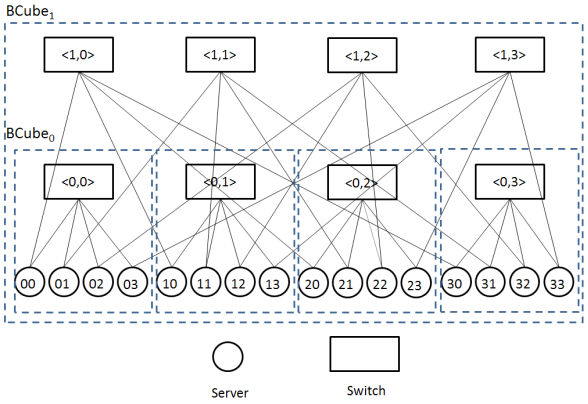


FIGURE 7: BCube network architecture in [12].

and applications, with the specific aim of addressing challenges in scalability (hundreds of thousands of servers), performance (high bandwidth and low latency), manageability (easy to operate and diagnose) and security (information isolation). Following is a brief introduction to three of these.

In our modular DCN architecture project, we designed a server-centric *BCube* architecture [12]. In BCube, servers with multiple ports connect to multiple layers of low-cost COTS

switches. BCube provides multiple parallel paths between any two servers and speeds up one-to-one, one-to-several, and one-to-all communication. BCube's network capacity for all-to-all data shuffling scales linearly with the number of servers. Due to its modular design, BCube can be extended to scale to networks that contain millions of servers. We were one of the first research groups publishing in data center networking. After being published in the September 2009 issue of ACM SIGCOMM, BCube has generated more than 150 citations.

ServerSwitch is a programmable and high-performance DCN platform [13]. It is designed to meet: (1) requirements for flexible packet forwarding as required by new DCN topologies, network virtualization, and multi-path routing designs; (2) real-time flow/congestion control for low-latency communication; and (3) in-network packet processing needs in new DCN applications. ServerSwitch takes advantage of the best of general purpose CPU and commodity switching ASIC. It leverages commodity switching ASICs for limited programmability yet high-speed packet forwarding, CPU for flexible and reasonable high-throughput packet processing; and PCI-E for high-throughput, low-latency interconnect between CPU and switching ASICs. ServerSwitch won the Best Paper Award at USENIX/ACM NSDI 2011. We believe ServerSwitch has the potential to disrupt the network device business.

Due to the sheer size of DCNs and rigid 24x7 operational requirements, DCN management and monitoring is a daunting task. As a first step towards automatic DCN management, we have helped the Microsoft Online Service Division (OSD) build *Pingmesh* to monitor network latency. With Pingmesh, every server is instrumented to launch TCP/HTTP pings to other servers. A centralized Pingmesh generator controls which server should ping which server, by considering network constraints and user requirements. Latency measurements are streamed to a network manager for real-time analysis and Cosmos, a distributed file system, for offline deep analysis. Pingmesh has been deployed in all OSD data centers, which include hundreds of thousands of servers; it now supports network health monitoring, capacity planning, and live-site diagnosis.

Distributed Systems Research

Distributed systems lie at the heart of cloud-based services, including Windows Azure, SQL Azure, Bing, Office 365, and Windows Live. The cloud-based service model introduces fundamental changes that force us to re-examine our current practices in designing, developing, testing, and deploying software. In the last six years, MSRA has invested heavily to build an area of expertise covering a wide spectrum of the service life cycle. We firmly believe that this area is of strategic importance to Microsoft and the industry in the shift from desktop to the cloud. Our research has not only established our thought leadership in the research community through a series of publications at top conferences, but has also begun to have an impact on Microsoft services through our strong partnership with product teams across Microsoft, spanning organizations such as OSD, Server and Tools business (STB), and Office 365.

PASS (Program Analysis on SCOPE Scripts) is an ongoing collaboration with the OSD Cosmos team that targets improving SCOPE script correctness and performance using program analysis techniques. SCOPE is the programming language and the runtime on Cosmos clusters of

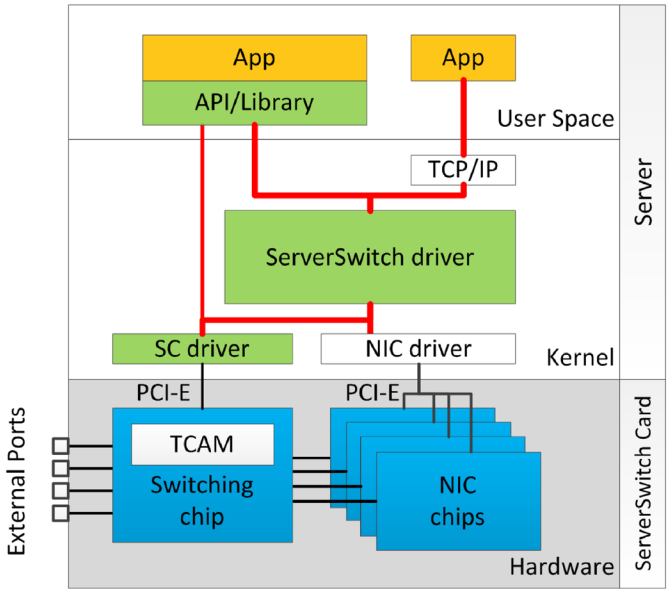


FIGURE 8: System architecture for ServerSwitch in [13].

tens of thousands of machines. We are currently focusing on three project aspects: SCA (Static Code Analysis) allows SCOPE users to discover potential run-time defects before submitting jobs to a Cosmos cluster; SUDO defines data properties related to data shuffling, exposing functional properties about user-defined functions for better optimization; and NEMO performs entire program analysis on a SCOPE script to achieve better performance. SCA has been integrated into the SCOPE IDE and released for use by thousands of SCOPE developers since October 2011.

Testing, which is crucial to the overall quality of cloud services, is particularly challenging in the cloud era due to faster service release cycles. Corner cases are more likely to surface in a real deployment under heavy use; it is often significantly more difficult and costly to investigate live-site issues, not to mention the irreparable damage to our reputation that each service disruption can potentially cause. It is therefore of paramount importance to expose and fix such corner-case bugs during the testing phase. Unit testing and isolated code coverage become vastly insufficient to ensure quality. Instead, complex interactions among modules/machines are often the source of problems; failures are the norm in systems with built-in, often error-prone recovery mechanisms. Non-determinism is abundant, making bug reproduction a huge pain point. These new challenges call for effective tools that are designed specifically for them.

We have developed Modist [15,16,17], a key enabling technology that can significantly improve our ability to test distributed systems in the following aspects: (1) to uncover hard-to-find distributed-systems bugs due to message interleaving, failures, and other non-determinism;

(2) to ensure stable failure repro in distributed systems; (3) to facilitate the effective root-cause analysis of complex issues. Modist's underlying technology has matured over four years' of research and has been both accepted in the research community and tested on real systems. Since early 2010, the joint incubation effort with the SQL Azure test team for use in testing production has helped us significantly improve the tool's practicality. Early experience has shown that Modist effectively reduces bug reproduce time from hours to minutes, while also assisting in root-cause analysis.

While most of our cloud-service efforts have focused on building functionality and features, it is our strong belief that every cloud service should come with a co-deployed infrastructure for live monitoring and diagnosis to enable fast response to incidents and anomalies. From a practical perspective, a high-quality monitoring and diagnosis system should ideally be: (1) real-time for fast decision making; (2) scalable for highly distributed services, (3) flexible to accommodate different modules and information sources; and (4) lightweight with the least interference to cloud services to guarantee the customer experience. This is a core competency that can significantly improve the perceived quality of cloud services.

CloudMeter is a monitoring and diagnosis infrastructure that is designed to meet these requirements by leveraging our G2 research project [18]. Our initial prototype was co-developed with the SharePoint Online Manageability team and is ready for internal testing. It validated CloudMeter's basic design and provides valuable experience towards real deployment. The CloudMeter project won the Golden Volcano Award at the Microsoft Services & Cloud Science Fair in February 2012.

Tiger is the code name for Bing's new index-serving platform, the platform that is responsible for finding and returning the most relevant documents on the web given a user query. Index serving is arguably the most complicated and critical infrastructure piece within a search engine, due to the huge amount of data involved, tight latency budget, extremely high query volume, and stringent reliability requirements. Index servers are usually running on a significant proportion of server machines. Therefore, efficient and effective index serving is very important to Bing's success.

The previous generation index server used by Bing put most index data in the main memory, and used the same data for both matching (i.e., finding the documents that contain the query words) and ranking (i.e., finding the best documents among all matching ones). Tiger leverages Solid State Disks (SSDs) in the index-serving platform. SSDs are faster than hard disks and cheaper than main memory. They present a new trade-off between performance and cost in the memory hierarchy. Since SSDs follow Moore's Law, we can expect to see better and cheaper SSDs in our datacenter servers in the near future. To best exploit the performance characteristics of SSDs, the Tiger team designed a novel two-layer index data structure. The new design uses different data for Matching and Ranking, and can thus significantly improve the index servers' efficiency due to the decoupling of the matching and ranking phases.

Tiger was first released for production in August, 2011. The first release provided significant performance gains compared to the previous generation index server platform with no

relevance degradation. It preserves most of the existing investment in the ranking infrastructure and the index preparation pipeline. The Tiger index server currently runs on thousands of machines, and already serves 100% of Bing's traffic. Not only does the Tiger architecture provide a significant efficiency gain, it also provides many new opportunities for improvements in efficiency and relevance going forward. The next versions of Tiger are expected to be significantly more efficient than the first release.

Epilogue

"Some people think research is all about technology," Rick often says. "It isn't. Research is all about people who have the ability to think and innovate. I think it's because we built that kind of organization with those kinds of values that we've been successful." We owe a lot of MSRA's success to Rick, not only for his technical influence and inspiration from Rick's seminal works described in this paper, but also for the research methodology and values he instills in us at Microsoft Research.

The first and foremost mission Rick set forth for Microsoft Research was focusing on fundamental research to advance the state of the art in any research area we explore. This exceptional foresight has allowed us to build an organization of high caliber; one that contributes to the community as well as providing a technical reservoir for Microsoft; experts at responding to the challenges of the ever-dynamic computer industry.

Not just satisfied with finding elegant solutions, Rick employs and encourages "*can-do*" and "*system building*" as his research philosophy. This philosophy has always been MSRA's proudest characteristic. This philosophy also helps promote a "*risk taking*" and "*fail fast*" culture within Microsoft Research with the "*trying something, seeing if it works, and if it doesn't, trying something else,*" attitude articulated by Rick. The "*system building*" practice has enabled Microsoft Research's contributions to many key Microsoft products over the last 20 years through effective technology transfers.

Microsoft Research Asia is extremely grateful for the opportunities given by Rick Rashid in the last 13 years—to push the boundaries of innovation, to use our imagination to turn ideas into reality, and to help build the lab into a world-leading research institution. We cannot possibly have made such attempts, let alone reached any such achievements without Rick's unparalleled leadership. We look forward to continuing this thrilling and rewarding journey with Rick for many years to come.

References

[1] Huang, Gregory T. "The World's Hottest Computer Lab." *MIT Technology Review*. June, 2004. MIT. http://www.technologyreview.com/Biztech/13616/.

[2] R.F. Rashid, "Towards a System for the Interpretation of Moving Light Display," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 2, no. 6, pp. 574-581, 1980.

[3] Yan Li, Tianshu Wang, Heung-Yeung Shum, "Motion Texture: A Two-Level Statistical Model for Character Motion Synthesis," ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH) 2002.

[4] W. Zhang, Q. Wang, and X. Tang. "Real time feature based 3-d deformable face tracking," In European Conference on Computer Vision (ECCV), pages 720-732, 2008.

[5] Avatar Kinect. http://www.xbox.com/en-us/kinect/avatar-kinect

[6] Microsoft LifeCam: http://www.microsoft.com/hardware/en-us/webcams

[7] Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, Weiwei Xu, "Sampling-based Contact-rich Motion Control," ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH) 2010.

[8] Haoda Huang, Jinxiang Chai, Xin Tong, Hsiang-Tao Wu, "Leveraging Motion Capture and 3D Scanning for High-fidelity Facial Performance Acquisition," ACM Transactions on Graphics, (Proceedings of ACM SIGGRAPH) 2011.

[9] Kun Tan, Jiansong Zhang, Ji Fang, He Liu, Yusheng Ye, Shen Wang, Yongguang Zhang, Haitao Wu, Wei Wang, and Geoffrey M. Voelker, "Sora: High Performance Software Radio Using General Purpose Multi-core Processors," NSDI 2009.

[10] Yong He, Ji Fang, Jiansong Zhang, Haichen Shen, Kun Tan, Yongguang Zhang, "MPAP: virtualization architecture for heterogeneous wireless Aps," ACM SIGCOMM Computer Communication Review, Volume 41 Issue 1, January 2011. SIGCOMM'10 best demo.

[11] Dina Katabi, "Technical Perspective: Sora Promises Lasting Impact," Communications of the ACM, 54(1), January 2011.

[12] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu, BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers, ACM SIGCOMM, August 2009.

[13] Guohan Lu, Chuanxiong Guo, Yulong Li, Zhiqiang Zhou, Tong Yuan, Haitao Wu, Yongqiang Xiong, Rui Gao, and Yongguang Zhang, ServerSwitch: A Programmable and High Performance Platform for Data Center Networks, NSDI 2011.

[14] Jiaxing Zhang, Hucheng Zhou, Rishan Chen, Zhenyu Guo, Xuepeng Fan, Haoxiang Lin, Jack Y. Li, Wei Lin, Jingren Zhou, Lidong Zhou: Optimizing Data Shuffling in Data-Parallel Computation by Understanding User-Defined Functions, NSDI 2012.

[15] Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long, Lintao Zhang, Lidong Zhou: MODIST: Transparent Model Checking of Unmodified Distributed Systems. NSDI 2009

[16] Huayang Guo, Ming Wu, Lidong Zhou, Gang Hu, Junfeng Yang, Lintao Zhang: Practical software model checking via dynamic interface reduction. SOSP 2011

[17] Zhenyu Guo, Xi Wang, Jian Tang, Xuezheng Liu, Zhilei Xu, Ming Wu, M. Frans Kaashoek, and Zheng Zhang: R2: An Application-Level Kernel for Record and Replay. OSDI 2008

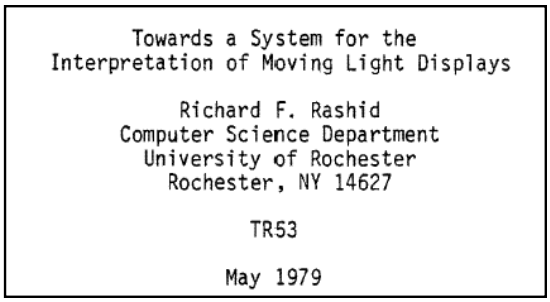
[18] Zhenyu Guo, Dong Zhou, Haoxiang Lin, Mao Yang, Fan Long, Chaoqiang Deng, Changshu Liu, Lidong Zhou: G2: A Graph Processing System for Diagnosing Distributed Systems, USENIX ATC 2011.

Moving in Mysterious Ways

Andrew Blake

Microsoft Research Cambridge
Rick Rashid Festschrift Colloquium
Friday, 9th March 2012

It is a great pleasure to be here to celebrate Rick's 60th birthday, along with his colleagues from the whole span of his career in computer science. For myself, I came on the scene relatively recently compared with the colleagues here at the colloquium who have worked alongside Rick for 30 years or more. I joined Microsoft Research 12 years ago, having previously been a university professor, and have hugely enjoyed the tremendously fertile and productive environment that Rick's vision and persistence have pulled together. It is a privilege to work in a lab like MSR which, almost uniquely amongst industry research labs, is fully signed up to the value of basic research, and its centrality in any company research environment that is actually sustainable. So congratulations to Rick, not only for the achievement of reaching 60, but also for his unmovable constancy and for keeping the faith, for over 20 years, that basic research should drive innovation in Microsoft's research labs.Rick's speciality has been mostly in computer systems research, whereas my own focus has been elsewhere, in computer vision. Apparently there is not so much of a link between our spheres of activity. But not all is what it seems. Back in the mists of time—the late 70s—Rick studied for his doctorate (figure 1) at the University of Rochester, and researched theories of the perception of "biological motion," moving dot videos of the human body, first invented by Johannson [1]. (The visible dots are placed typically on the joints of the body as in figure 2.) The remarkable thing about these displays is that they are rather a pure form of motion stimulus, in that a single, static frame from such a movie conveys no percept,


Towards a System for the
Interpretation of Moving Light Displays

Richard F. Rashid
Computer Science Department
University of Rochester
Rochester, NY 14627

TR53

May 1979

Figure 1.Technical report summarising Rick's thesis work on motion perception

FIGURE 2: A birthday gift from MSR Cambridge—apparatus for motion experiments.

appearing just as a jumble of dots. Once the movie ***plays***, the percept of the moving human figure immediately springs into life. In this respect it is the analogue, for motion, of the famous "random dot stereograms" of Julesz [2], which play the same role for stereo vision: random patterns that are meaningless when presented to just one eye, but when a stereo-pair is viewed, a full surface relief appears. Biological motion is a fine playground for researching the properties of motion perception, in a way that is rather independent of other channels for perception of form. So Rick made an excellent choice there.

Computer vision is hard, and this had become apparent when, a decade before Rick was writing his thesis, scientists at MIT had set the "vision" problem as a summer project for some students [3]. Such was the level of optimism at the outset, that goals were set for July, and then further goals, covering a more general setting for the problem, were to follow in August. In the event, the vision problem proved much harder, and we are still grappling with it almost 50 years later. Partly this is because of the generality of the problem and the deceptive ease with which our own brains solve it. Figure 3 illustrates this. In attempting to interpret the image in the centre, our brains seem effortlessly to perceive the outline on the right. And yet, when an honest attempt is made to apply standard signal processing to the image, in the hope of extracting the outline of the hand as a contour of high contrast in the image intensities, a typical result would be the one shown on the left. The outline is present, but rather broken-up and imperfect. It is also masked by apparently irrelevant contours, but generated in fact by texture markings and shadows that really are present in the original hand image. Of course by studying moving dot displays, Rick tastefully finessed some of these complexities, to enable the research to concentrate on one particular aspect of the vision problem, namely the tracking of moving features. The "Lights" system that he developed showed how a combination of prediction over time, and global error minimization over space, could successfully track the motion of the human body.

FIGURE 3: Why is vision hard?

In addition to being a useful scientific playground, biological motion is an important application in its own right. Commercial systems like Vicon use multiple cameras to track the motion of humans whose bodies have been instrumented with markers on the joints. Tracking and reconstruction software records the 3D configuration of the body over time. This has been used to analyse the gait of children in need of corrective surgery. A more recent application is for virtual reality in the movie industry, where it is now used routinely for

special effects. Partly inspired by systems like Vicon, and the awkwardness of having to place markers on the body, my colleagues at the University of Oxford and I started some years ago researching markerless visual tracking systems. We generated one of the earliest demonstrations of real-time hand-tracking [4], and later an experimental system for tracking the whole body [5]. This line of work continued at Microsoft Research [6], so that when Microsoft Xbox had produced their "Natal" prototype of what we now know as the Kinect camera (figure 4), and wanted to know how to use it best to track the human body, we had already thought a good deal about the problem. Of course Natal was a game-changer, because it was a 3D camera, which immediately transformed the first part of the problem—peeling the subject away from its background—something that is hard to do with a monocular camera [7]. In the meantime, while this line of research had gone a little quiet at Microsoft, a team at Toshiba had taken it much further [8]. Whereas we had generated a "flip-book" of a few dozen pages, each containing an "exemplar"—a cartoon-like pose of the body—Toshiba had scaled this up to around 50,000 pages. Tracking then consisted of rushing backwards and forwards in the book, making probabilistic transitions between pages, to keep up with the pose presented in successive video frames. When my colleagues and I came to consider the problem again in the context of Kinect, we realised that 50,000 exemplars would not be sufficient to cover all of the poses of the human body that would be encountered in the relatively unconstrained setting of game-play. While we were pondering what to do about this, the newly-hired MSR researcher Jamie Shotton was thinking along quite different lines. He and a few colleagues had been working speculatively on object recognition [9], and this led him to what seemed a very surprising approach at the time—labelling every pixel of the foreground of the Kinect depth-image as belonging to one of 31 possible body parts (figure 5).

Figure 4: The Kinect 3D camera

Figure 5: a) Depth image from the Kinect 3D camera. b) Every pixel labelled according to part of the body

To cut a long story short, this is the basis of the approach adopted for the machine learning behind the human body recognition in Kinect. It is described in detail elsewhere [10]. Curiously, the Xbox team continued to call this the "Exemplar" component of the software, long after the use of exemplars had been abandoned—a historical artefact that simply wouldn't die.

Well that completes a story that links all the way through from Rick's earliest research work to one of Microsoft's most recent products. All that remains is to congratulate Rick on his birthday, to wish him many more happy and prosperous years, and to suggest that this might

be a good moment to return to his earliest research topic. The problem of human body motion analysis is by no means closed, and now Rick has exactly the experimental apparatus (figure 2) that he needs.

References

[1] G. Johansson (1973). "Visual perception of biological motion and a model for its analysis." Percept. Psychophys. 14 (2): 201–211.

[2] B. Julesz, (1971). Foundations of Cyclopean Perception. Chicago: The University of Chicago Press.

[3] S. Papert (1966). The Summer Vision Project. MIT AI Memo 100.

[4] A. Blake and M. Isard (1994). 3D position, attitude and shape input using video tracking of hands and lips. Proc. ACM SIGGRAPH Conference, Orlando, USA, 185-192

[5] J. Deutscher, A. Blake and I. Reid (2000). Articulated Body Motion Capture by Annealed Particle Filtering Proc. Conf. Computer Vision and Pattern Recognition, 2, 126-133.

[6] K. Toyama and A. Blake (2002). Probabilistic tracking with exemplars in a metric space. Int. J. Computer Vision, 48, 919.

[7] V. Kolmogorov, A. Criminisi, A. Blake, G. Cross and C. Rother (2006). Probabilistic fusion of stereo with colour and contrast for bi-layer segmentation. IEEE Trans. Pattern Analysis and Machine Intelligence, 480-1492

[8] R. Okada and B. Stenger (2008). A Single Camera Motion Capture System for Human-Computer Interaction. Trans. IEICE, Vol. E91-D, No.7, pages 1855-1862

[9] J. Shotton, J. Winn, C. Rother, and A. Criminisi (2006). TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Recognition and Segmentation. European Conf. Computer Vision.

[10] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M.Finocchio, R. Moore, A. Kipman and A. Blake (2011). Real-Time Human Pose Recognition in Parts from a Single Depth Image. In IEEE Proc. Computer Vision and Pattern Recognition.

From Phase Transitions to Network Algorithms: A Personal Research Journey on Rick's Long Leash

Christian Borgs and Jennifer Chayes

It is with great pleasure that we dedicate this to Rick Rashid on the occasion of his 60th birthday. Rick's vision and leadership have had a profound influence on computer science and technology. Under Rick's stewardship, Microsoft Research has become the premier institution in the world in computer science research, broadly defined. MSR has taken the best aspects of the legendary labs of the past—free, open and transformative research in the tradition of Bell Labs and Xerox PARC—and coupled these with an understanding of the need to bring many of the fruits of the research back to the company that sponsors it. As Rick explained to us when we joined the lab fifteen years ago, the key is to identify the correct point at which to manage the research pipeline. We hire the best and the brightest in areas that we believe will transform computer science and technology, and then we step out of the way and see what happens. Rather than second-guessing the experts we hire, we let them follow their passion and do research that advances the state of the art in their fields. We manage the research pipeline at the end rather than the beginning, seeing what comes out of the research, and then helping to make the connections that ensure that Microsoft will benefit from it. It's a simple but incredibly powerful vision. Just as Rick explained it to us in 1997 when we joined MSR, we have explained it to numerous researchers over the years. We still marvel at the results. It is a testimony to Rick's leadership that so many of the leaders of MSR reflect his vision and style of leadership in their endeavors.

Many of our lessons from Rick over the years have been on the subject of leadership, and indeed there is no way that we could have founded Microsoft Research New England without the model Rick provided to us. But this piece will be more about our personal journey, as researchers rather than as research leaders in MSR. When Rick, Nathan Myhrvold and Dan Ling first hired us in 1997, we were certainly outliers in MSR: mathematical physicists who rigorously proved theorems about phase transitions in systems like exotic magnets. These endeavors had brought us to a point of not-very-close approach to computer science, namely

studying phase transitions in intractable constraint-satisfaction problems. We were studying abstract systems with resources and constraints among those resources, and searching for solutions which optimized or nearly optimized the number of constraints satisfied. These systems underwent transitions as the ratio of the number of resources to the number of constraints varied, with it being possible to satisfy all the constraints when resources were relatively abundant, and not possible to satisfy them when the system was dominated by constraints. Somewhat surprisingly, the nature of this transition was mathematically remarkably similar to phase transitions in certain exotic magnets called spin glasses. Clearly this was not a particularly close approach to computer science or technology!

Our Journey

Just as we've learned to give our researchers wide berth in their research choices, MSR under Rick gave us complete freedom. For some time, we used this freedom to continue to do work in mathematical physics. But we also realized that graphical representations of the random magnets we were studying were in a fact a good framework for modeling and understanding growing networks—of the type beginning to become popular in the study of the Internet and the World Wide Web. Indeed, over the years, we've constructed and established many properties of probabilistic and game-theoretic models of the Internet, the World Wide Web, and a host of social networks [1], [2], [3], [4].

In major work begun with mathematicians in the Theory Group in MSR Redmond, we've even constructed an abstract theory of growing networks and their limits [5], [6], [7], and in particular how one would test or sample from those networks to determine various properties [8].

Constructing models of growing networks is just a first step in their study. Another natural step is to study processes on these networks. We've done this in the context of both the spread of viruses and worms on networks [9], [10], and the spread of information on networks [11].

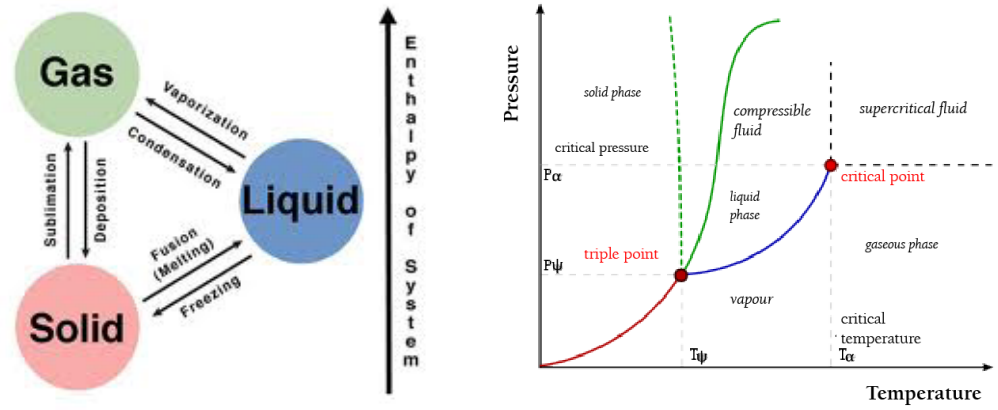
The next natural step in the study of networks is to develop network algorithms—either algorithms on networks, or algorithms to reconstruct networks from partial data. Among the works we've done in this regard are algorithms for crawling and spam detection on networks [12], [13], recommendation system algorithms on social networks [14], [15], sub-linear time algorithms for Pagerank and for finding high-influence nodes [16], algorithms for solving matching problems [17], and finally network reconstruction algorithms [18], [19].

In this short piece, we'll look at the development of a class of network algorithms which grew out of the study of phase transitions in combinatorial optimization problems, and then focus on one particular algorithm, for finding so-called "Steiner trees" [20]. We'll show that, in addition to solving some more standard network problems—like finding the best paths for multicasting—this algorithm has surprising applications to gene regulatory network reconstruction, in particular to identifying drug targets for cancer [18], [19].

Phase transitions: From Fluids to Random k-SAT

A phase transition is an abrupt change in the state of a material or system as an external parameter is varied. Canonical examples of phase transitions include the liquid-gas and liquid-solid transitions in response to changes in temperature and pressure.

Mathematically, the phase transition is characterized by non-analyticities in various thermodynamic quantities describing the properties of the system.



Somewhat surprisingly, phase transitions can also occur in combinatorial optimization problems of interest to computer science. A canonical example is the k-satisfiability (k-SAT) problem, which models systems in which there are resources and constraints of a particular form: an instance of the random k-satisfiability problem is a k-CNF formula with m clauses; each clause contains an "or" of k variables and their negations drawn uniformly from among n variables; and the m clauses are connected by "ands" so that all must be satisfied simultaneously for the formula to be true. The following is an example of a Boolean CNF formula with k = 3 literals per clause and m = 3 clauses:

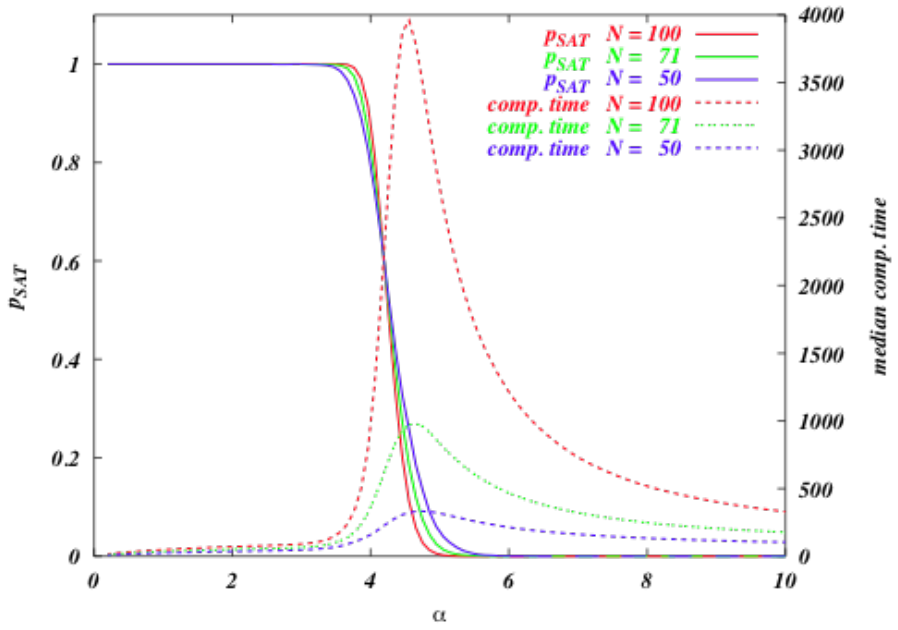
$$(x_5 \vee \bar{x}_7 \vee x_{12}) \wedge (x_{21} \vee x_{28} \vee \bar{x}_{30}) \wedge (x_9 \vee \bar{x}_{11} \vee \bar{x}_{12})$$

For the random k-SAT problem, we form a random k-CNF formula $F_k(n, m)$ by choosing m clauses uniformly at random from the set of all $\binom{n}{k}2^k$ k-clauses on the variables $x_1, \ldots, x_n$. We ask whether $F_k(n, m)$ has a satisfying truth assignment, and how does the probability of this vary as a function of c = m/n? When m/n is small, so that there are many variables relative to constraints, the formula is satisfiable with high probability. When m/n is large, with high probability the formula is over-constrained and hence unsatisfiable. What is interesting is that there is a sharp transition at a particular (k-dependent) value of m/n,

$$\lim_{n\to\infty} \Pr[F_k(n, cn) \text{ is satisfiable}] = \begin{cases} 1 & if \quad c < c_k \\ 0 & if \quad c > c_k \end{cases}.$$

Moreover, many of the properties of that transition are analogous to the phase transition in exotic disordered magnets, called spin glasses.

Even more interestingly, there is a dramatic decrease in the performance of conventional algorithms (e.g., Davis-Putnam-type algorithms) near the phase transition [21], as illustrated below.



In fact, the difficulty of solving these problems with conventional algorithms turns out to be related to the structure of the solution space: roughly speaking, near the transition, it is difficult to get from one solution to another via local moves. Hence there is a need to develop new algorithms, of a more distributed nature, which simultaneously probe many very different possible solutions. Physicists suggested heuristics based on this reasoning which led to the development of a new class of algorithms which they call survey propagation [22], [23].

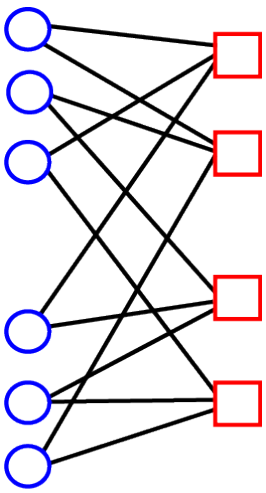
New Distributed Algorithms

The survey-propagation algorithms turn out to be generalizations of the belief-propagation algorithm, first suggested by Pearl in the context of message-passing algorithms on graphical models [24]. Belief-propagation algorithms have been widely used in many applications, e.g., coding theory and vision. However, although belief-propagation algorithms have been widely deployed and very successful in addressing many applications, there is remarkably little rigorous work proving the convergence of these algorithms when the underlying graphical model is not a tree.

Let us formulate a constraint-satisfaction problem and show how to approach it with a belief-propagation algorithm. Introduce a probability distribution over assignments $\{x_i\}$:

$$P(\{x_i\}) = \frac{1}{Z}\, exp(-\beta\, E(\{x_i\}))$$

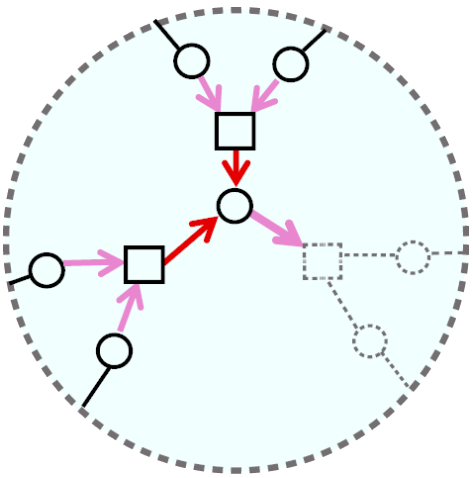
where the "cost" $E(\{x_i\})$ is the number of UNSAT clauses. To describe the new class of algorithms, it is convenient to introduce the so-called "*Factor graph.*" This graph is a graph on a set of nodes consisting of one node for each variable x_i (called variable nodes) and one node for each clause C in $F_k(n, m)$ (called constraint nodes). The factor graph $\mathcal{F}$ is obtained by drawing an edge between i and C if x_i is a variable in C.



The *belief propagation* (BP) algorithm approximately calculates the probability distribution P and hence the satisfying assignments by passing messages along the edges of the factor graph $\mathcal{F}$. More precisely, consider the following marginal:

- $\mu_{i\to C}(x_i)$ marginal of x_i where C is left out
- $\mu_{C\to i}(x_i)$ marginal of x_i where all constraints containing x_i except for C are left out.

Belief Propagation is an approximation algorithm for calculating these marginals via *message passing* on the factor graph, as shown in the following picture:



Survey Propagation (SP) [22], [23] is a generalization of BP involving distributions over marginals, instead of marginals themselves. Near the phase transition of random k-SAT, SP appears to work orders of magnitude better than modified Davis-Putnam and other standard algorithms. Moreover, it is highly parallelizable, and hence works well on many cores.

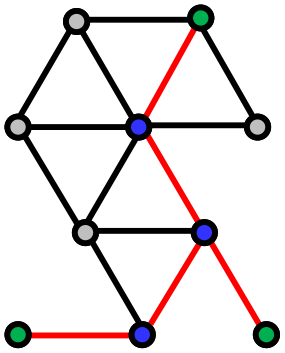
The Steiner Tree Problem

Let us begin with a particularly simple example of the Steiner tree problem involving only three fixed nodes. Assume we have three equidistant planets, each pair separated by one light year, and that we need to get some resource from one to the two others, using as little fuel as possible. The question is what path we choose, and what is the cost of that path. The solution is simply the minimum cost spanning tree, which in this case costs two light years' worth of fuel.

Now consider the variant in which the Starship Enterprise is in the vicinity of the planets, say in the gravitational center right in the middle of the 3 planets. The minimum spanning tree is now clearly a star graph with the Starship Enterprise as the center, and the cost can easily be calculated to be 3/(2 sin 60°)=1.73.

Clearly, the existence of an extra node, in this case the Starship Enterprise, has led to a lower cost solution. Nodes which lead to lower cost solutions are called *Steiner nodes*.

The Steiner tree problem can also be generalized to networks. In this case, we are given:



- Graph $G = (V, E)$
- Set of "terminals" $S \subseteq V$
- Costs $\{c_{ij}\}_{ij \in E}, c_{ij} \geq 0$.

The problem is to find a tree $T \subseteq G$ containing all terminals, with minimal cost

$$C(T) = \sum_{ij \in T} c_{ij}$$

The extra nodes in T (i.e., those not in S) are called Steiner nodes. The problem is illustrated in the figure above, where the terminals are marked in green. In this example, the Steiner tree is given by the red edges, and the extra Steiner nodes are the blue nodes.

There are many applications of the Steiner tree problem in networks, e.g., broadcast algorithms, where the root is the server, and all other terminals are customers.

Another variant is the so-called *Prize-Collecting Steiner Tree* (PCST). Here we:

- Relax the constraint that all terminals need to be included ("served")
- Instead, collect prize π_i for each terminal included in the tree

In other words, we need to find the tree $T \subseteq G$ which minimizes the cost:

$$C(T) = \sum_{ij \in T} c_{ij} - \sum_{i \in T} \pi_i$$

One of our main contributions to the Steiner tree problem was to devise a BP algorithm for finding Steiner trees on networks [20].The difficulty here was that, in its conventional representation, the Steiner tree problem has a global connectivity constraint, namely that the solution be a (connected) tree. But BP algorithms can only be realized locally, via message passing among adjacent sites, which typically does not enforce global constraints.

Our solution was to devise a *new local representation* for the Steiner tree problem, which manages to enforce the global constraint via local constraints. We did this by choosing a root r among the terminals, and defining two local variables d_i, p_i for each node i in G, where:

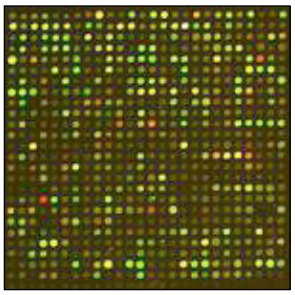
- d_i is the distance from i to the root r in T
- p_i is a pointer pointing to the parent of i in T

In this new representation, all constraints are local, simply relating the variables d_i, p_i for adjacent nodes—which allows us to find minimum cost Steiner trees using BP.

It turns out that, in practice, our algorithm outperforms the best previous algorithms for almost all benchmark problems in the so-called "SteinLib" library of problems by several orders of magnitude [20].

Applications to Gene Regulatory Networks

One of the nicest applications of our BP algorithms for Steiner trees is to the reconstruction of gene regulatory networks, and to the identification of previously unidentified proteins of importance in particular gene regulatory pathways. So called "gene chips" give us sets of proteins which are "differentially expressed" under given experimental conditions.



The problem is that some proteins integral to the relevant pathways may not be expressed at a different level than the background. Our idea is to find these missing proteins as the Steiner nodes in a prize-collecting Steiner tree (PCST) problem with

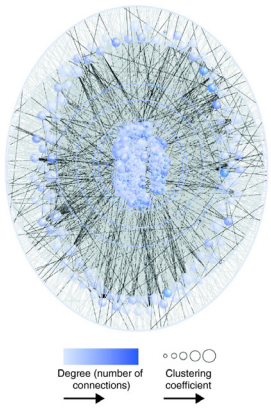
- prizes $\pi_i = -\log(p_{value}(i))$
- costs $c_{ij} = -\log(\Pr\{\text{Proteins } i \text{ and } j \text{ interact}\})$

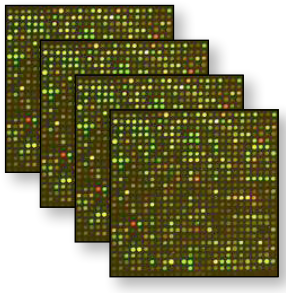
where the prizes vary with the given experimental conditions and the costs c_{ij} are determined from known protein-protein interactions of the given organism.

The first problem to which we applied this was a yeast pheromone response pathway [18]. We used the known yeast protein signal transduction network:

- 4689 Proteins
- 14928 Protein-Protein interactions

which gives a set of weights $\{c_{ij}\}$ for relevant protein-protein interactions in yeast.





We also considered 56 large-scale gene expression data sets used to reconstruct the yeast pheromone pathway. For each data set, we get a:

- set of prizes π_i

Using the weights and the prizes, we constructed 56 solutions to the PCST problem, and then merged these 56 trees to get one network.

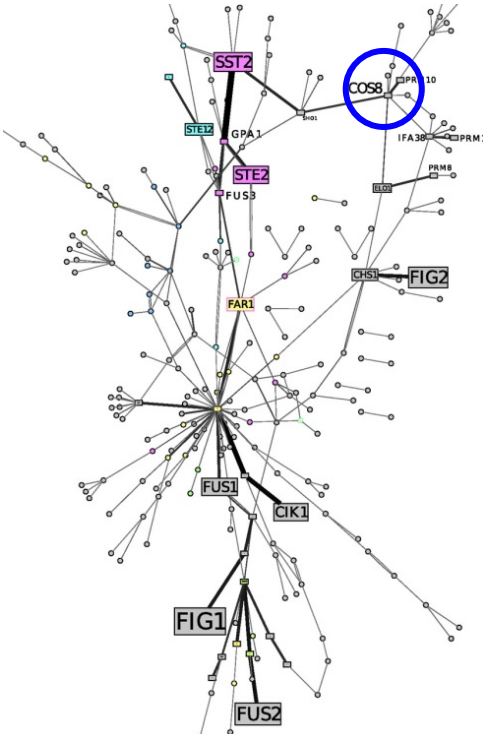
The resulting network contained two types of proteins:

- Proteins differentially expressed in pheromone response and previously discovered by transcriptomic studies (terminals)
- Proteins not differentially expressed but bridging between different sub-networks ("Steiner proteins")

The question that we asked is whether these Steiner proteins are important in the yeast pheromone response pathway.

Consider our resulting network right, on which we have indicated one of the Steiner proteins COS8:

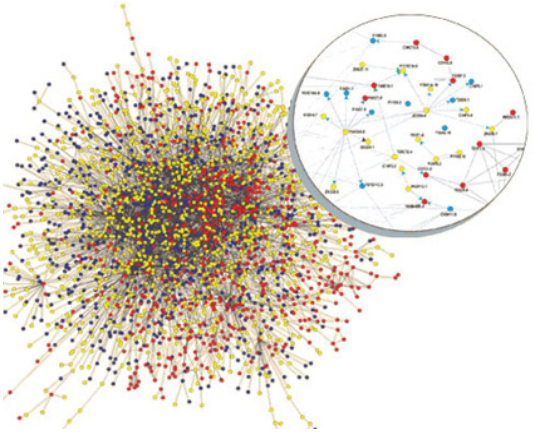
In order to experimentally test whether the COS8 protein was important to the pathway, we (actually, our biologist collaborators) knocked out the corresponding gene in yeast, and observed that the pheromone response pathway fell apart in the resulting organisms. We took this as "experimental validation" of the significance of our findings.



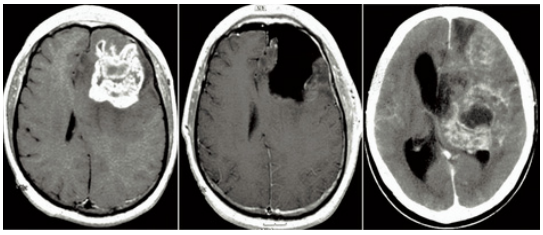
Next we turned to the reconstruction of gene regulatory networks in humans. The human interactome is much larger than the yeast interactome. Currently, it contains:

- 24,039 proteins
- 234,455 protein-protein interactions

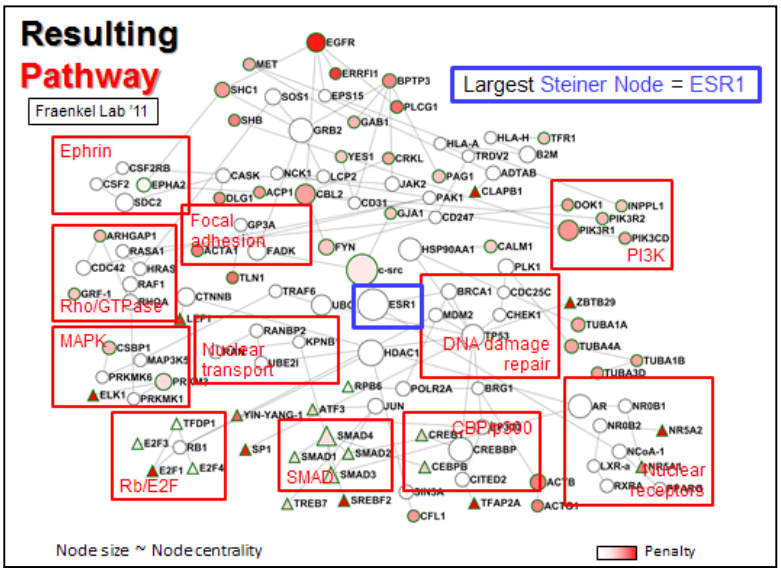
Our BP algorithm is the only algorithm that appears to work in practice; all known LP (linear programming) algorithms fail on human data sets. We are currently involved in using our algorithm on several problems in the so-called Cancer Genome Atlas. For these problems, the Steiner nodes we find represent potential drug targets for cancer.



Glioblastoma is a particularly virulent form of brain cancer. It is the deadliest human cancer, with no good known drug targets. Interestingly, glioblastoma is much more common in men than in women (about 80% of the patients are male).

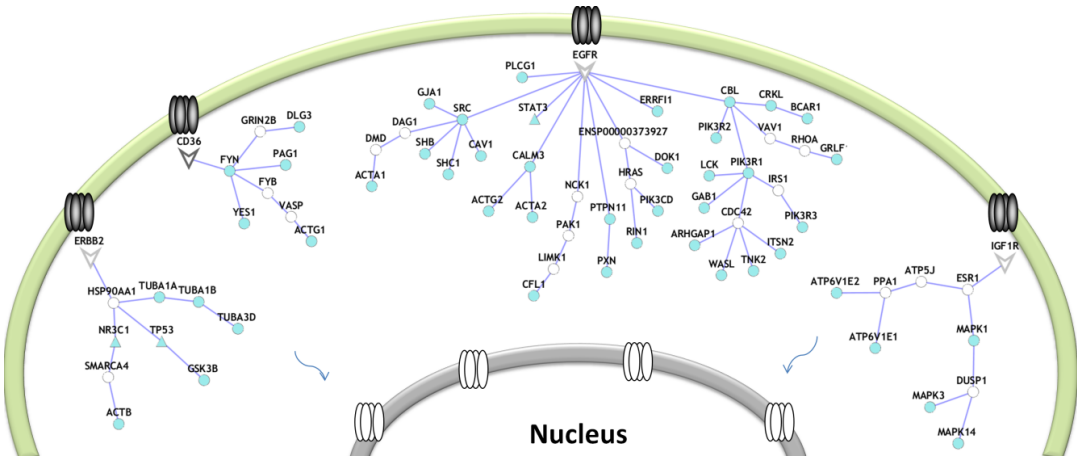


The Fraenkel Lab at MIT used our prize-collecting Steiner tree BP algorithm to find glioblastoma networks for hundreds of individual patients & merged solutions to get one pathway, illustrated below.



The most important Steiner node on this network was the ESR1. This is an amazing result—ESR1 is the estrogen receptor. Hence this is the first explanation of a sex link in glioblastoma! The Fraenkel lab went on to test whether ESR1 was really significant by treating glioblastoma cells in culture with estrogen. These cells fared worse than the control culture, and hence gave experimental validation of the findings.

We have since gone on to study, jointly with the Fraenkel lab, the problem of finding *Steiner forests*, representing disconnected pathways in cancer genomics. In particular, in the case of glioblastoma, we have found Steiner forests rooted at different cell membrane receptors [19], as illustrated below.



We intend to continue applying our BP algorithms to the Steiner tree and Steiner forest problems in various other cancers, including breast, ovarian and prostate cancers.

To recall a memorable phrase from Rick's youth and ours: "*What a long, strange trip it's been …*"

Bibliography

[1] B. Bollobas, C. Borgs, J. T. Chayes and O. Riordan, "Directed scale-free graphs," Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 132-139, 2003.

[2] C. Borgs, J. T. Chayes, C. Daskalakis and S. Roch, "First to market is not everything: an analysis of preferential attachment with fitness," Proceedings of the 39rd annual ACM Symposium on the Theory of Computing (STOC), pp. 135-144, 2007.

[3] N. Berger, C. Borgs, J. T. Chayes, R. D'Souza and R. D. Kleinberg, "Emergence of tempered preferential attachment from optimization," Proceedings of the National Academy of Sciences (PNAS), vol. 104, pp. 6112-6117, 2007.

[4] C. Borgs, J. T. Chayes, M. Mahdian and A. Saberi, "Exploring the community structure of newsgroups," Proceedings of the 10th ACM SIGKDD International Conference on Knowledge, Discovery and Data Mining (KKD), pp. 783-787, 2004.

[5] C. Borgs, J. T. Chayes, L. Lovasz, V. Sos and K. Vesztergombi, "Convergent sequences of dense graphs I: Subgraph frequencies, metric properties and testing," Advances in Math, vol. 219, pp. 1801-1851, 2008.

[6] C. Borgs, J. T. Chayes, L. Lovasz, V. Sos and K. Vesztergombi, "Convergent sequences of dense graphs II: Multiway cuts and statistical physics," Ann. of Math., 2012.

[7] C. Borgs, J. T. Chayes, J. Kahn and L. Lovasz, "Left and right convergence of graphs with bounded degree," Random Structures and Algorithms, 2012.

[8] C. Borgs, J. T. Chayes, L. Lovasz, V. Sos, S. B. and K. Vesztergombi, "Graph limits and parameter testing," Proceedings of the 38rd Annual ACM Symposium on the Theory of Computing (STOC), pp. 261-270, 2006.

[9] N. Berger, C. Borgs, J. T. Chayes and A. Saberi, "On the spread of viruses on the Internet," Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithm (SODA), pp. 301-310, 2005.

[10] C. Borgs, J. T. Chayes, A. Ganesh and A. Saberi, "How to distribute antidote to control epidemics," Random Struct. Algorithms, vol. 37, pp. 204-222, 2010.

[11] C. Borgs, M. Brautbar, J. T. Chayes, S. Khanna and B. Lucier, "The power of local information in social networks," Preprint, 2012.

[12] R. Andersen, C. Borgs, J. T. Chayes, J. E. Hopcroft, V. S. Mirrokni and S. H. Teng, "On the stability of web crawling and web search," ISAAC, pp. 680-691, 2008.

[13] R. Andersen, C. Borgs, J. T. Chayes, J. E. Hopcroft, K. Jain, V. Mirrokni and S. H. Teng, "Robust PageRank and locally computable spam detection features," AIRWeb, pp. 69-76, 2008.

[14] R. Andersen, C. Borgs, J. T. Chayes, U. Feige, A. Flaxman, A. Kalai, V. Mirrokni and M. Tennenholtz, "Trust-based recommendation systems: An axiomatic approach," Proceedings of the 17th international conference on World Wide Web (WWW), pp. 199-208, 2008.

[15] C. Borgs, J. T. Chayes, A. Kalai, A. Malekiany and M. Tennenholtz, "A novel approach to propagating distrust," Proceedings of the 6th International Workshop on Internet and Network Economics (WINE), pp. 87-105, 2010.

[16] C. Borgs, M. Brautbar, J. T. Chayes and S. H. Teng, "Sublinear time algorithm for PageRank computations and related applications," 9th Workshop on Algorithms and Models for the Web Graph (WAW)., 2012.

[17] M. Bayati, C. Borgs, J. T. Chayes and R. Zecchina, "Belief-Propagation for weighted b-matchings on arbitrary graphs and its relation to linear programs with integer solutions," SIAM Journal of Discrete Mathematics, vol. 25, pp. 989-1011, 2011.

[18] M. Bailly-Bechet, A. Braunstein, J. T. Chayes, A. Dagkessamanskaia, J. Francois and R. Zecchina, "Finding undetected protein associations in cell signaling by belief propagation," Proceedings of the National Academy of Sciences (PNAS), vol. 108, pp. 882-887, 2011.

[19] N. Tuncbag, A. Braunstein, A. Pagnani, S. C. Huang, J. T. Chayes, C. Borgs, R. Zecchina and E. Fraenkel, "Simultaneous reconstruction of multiple signaling pathways via the prize-collecting Steiner forest problem," RECOMB, 2012.

[20] M. Bayati, C. Borgs, A. Braunstein, J. T. Chayes, A. Ramezanpour and R. Zecchina, "Statistical mechanics of Steiner trees," Physical Review Letters, vol. 101, p. 037208, 2008.

[21] S. Kirkpatrick and B. Selman, "Critical behavior in the satisfiability of random Boolean expressions," Science, vol. 264, pp. 1297 - 1301, 1994.

[22] M. Mezard and R. Zecchina, "The random K-satisfiability problem: from an analytic solution to an efficient algorithm," Phys. Rev. E, vol. 66, pp. 505-543, 2002.

[23] A. Braunstein, M. Mezard and R. Zecchina, "Survey propagation: an algorithm for satisfiability," Random Structures and Algorithms, vol. 27, pp. 201-226, 2005.

[24] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," AAAI-82: Second National Conference on Artificial Intelligence, p. 133–136, 1982.

The Importance of Dog Food

Peter Lee

I first met Rick Rashid when I joined the faculty at Carnegie Mellon University in 1987. In those days, we used many different kinds of computing technologies to get our research and teaching done. We kept files and programs in a shared distributed file system and searched for them by sending queries to a system that used a crawler to index their contents. We communicated with each other by sending attachments in multimedia-rich emails and via instant messaging. And our computers ran on operating systems based on a "microkernel" concept, essentially the prototype for dozens of subsequent research and commercial operating systems, including Apple's OS X and iOS.

I suppose that many of the other papers in this volume will give some of the history and technical detail of these and other technologies of the day. What I think is important to understand, however, is the following: while we might take these technologies for granted today, in the 1980s and early 1990s only a few places—some top universities and industrial research labs—were developing such things. And in the Carnegie Mellon Computer Science Department, everyone was obligated to use these technologies for their everyday work. I vividly remember settling in to my new office in Wean Hall and getting a workstation running some flavor of the Mach operating system from Rick's research group. Was Mach actually usable for everyday work? It actually did quite well, considering how experimental it all was and the fact that it was being developed in large part by students. But whatever minor inconveniences might come from using research software for everything, I joined all of the other faculty members and graduate students in the department in "eating our own dog food." And I did so with glee. Living with all of these technologies, which obviously someday were going to be used by everyone, made me and all of my colleagues feel like we were living in the future.

My own research at the time was in formal semantics and compiler design. Having research interests so far from Rick's meant that I was never lucky enough to collaborate on a research

project with him before he left CMU to start Microsoft Research in 1991. But the fact that Rick helped to create such a strong dogfooding culture meant that I was able to gain some understanding of the core research issues across a wider swath of computer science than if I had just stuck to my own narrow research domain and used only commercially available computing infrastructure. This ultimately led me to collaborate with a number of faculty members, and most notably Eric Cooper, on research that investigated the value of typed programming languages in the development of networking software and, later in web servers. One result of this was the Fox Project [1].

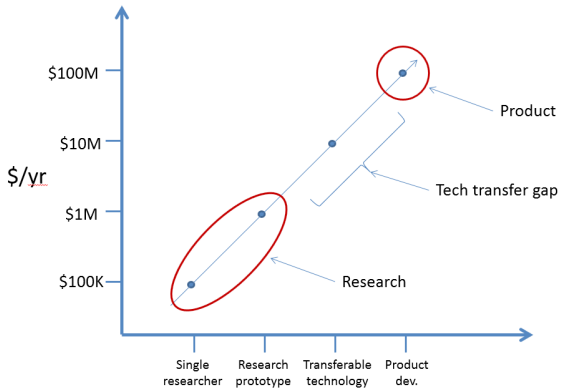
Later, after Rick had gone to Microsoft, new faculty members joined the department, and some of them came to advance the ideas started by Rick. They had a profound impact on my research. Of particular note were Brian Bershad and then Steve Lucco, who introduced me to different facets of the problem of how to get the functionality and performance of putting user code in the kernel address space, while at the same time avoiding the obvious risks to reliability and security. This research problem greatly motivated me. I thought that the kinds of programming language technologies I most believed in should provide good solutions to these problems. But there was a big problem: concepts such as advanced type systems, automatic garbage collection, dynamic compilation, and so on, were, at the time, restricted to fairly high-level languages, and not available to systems-level languages like C or even assembler.

After years of work, we made a number of significant advances. Greg Morrisett, Bob Harper, David Tarditi, Perry Cheng, and others (including me) developed the notions of "typed intermediate languages" and "type-directed compilation" [2]. And then in 1996 George Necula and I published our first paper on the concept of "proof-carrying code" and demonstrated how user-written packet filters could be safely run in the kernel of a microkernel operating system [3]. I think it's fair to say that this has been the most important research contribution of my career, and I think it is notable that it traces its roots directly back to Rick Rashid's early work on microkernels.

But let me get back to the central topic of dog food, because it is in the idea of creating a dogfooding culture that we can find a lesson about how to manage a research organization. Another key figure in those early days at CMU was Duane Adams. When I first met him in 1987, he introduced me to the idea of technology transfer, or "tech transfer" for short. Of course, nowadays everyone talks about tech transfer, but in 1987 this was still a pretty new topic. The Bayh-Dole Act that created incentives for universities to own intellectual property and seek licensing deals was only a few years old at that time, and tech transfer offices at universities were still just emerging.

Today, of course, tech transfer continues to be one of the most vexing challenges for any research organization that cares about making an impact in the real world (and I think most do). Duane, around 1989 or so, introduced me to a framework for thinking about this. This framework, which uses Rick's work on Mach as the key example, is something that stuck with me. I've used it at DARPA and continue to use it to this day in my management of the Microsoft Research lab in Redmond.

The figure below shows a graph, where points along the x-axis depict increasing levels of technology maturity and the y-axis shows approximate (annual) cost of research.



At the point closest to the origin, we have the research conducted mainly by a single researcher for a year. One can think of this as the baseline cost of a single research paper published in a conference or journal. The example Duane Adams had used was Rick Rashid working on the RIG system at Rochester, prior to his arrival at CMU. The work on the Mach operating system, then, takes us to the next stage, which one can think of as a team of researchers, including faculty, graduate students, research programmers, and the like, working together to develop a research prototype—like Mach in 1987. While the single-researcher activity costs on the order of $100K per year, a larger project like Mach, aimed at producing a distributable prototype, is closer to $1M per year.

At the very end of the spectrum is product development, which is a huge undertaking. In a conversation many years later with Avie Tevanian when he was leading the development of OS X, he mentioned that the total costs exceeded many tens of millions of dollars.

I have found this graph to be quite revealing for me personally, as well as a useful tool for explaining to business leaders and policy makers why tech transfer can be so difficult. It is also useful for explaining this to researchers. Research, being on the left end of the scale and relatively cheap, means that many "bets" can be placed and lots of ideas tried out. Many will fail but generally we will learn from both successes and failures and so it is ok. Product development, on the other hand, is a huge investment. Most companies can afford to make only a very small number of them. Furthermore, while the graph shows costs, in fact there is a huge amount of intellectual investment in product development. As the old saying goes, invention is only 1% inspiration and 99% perspiration. Interestingly, these percentages map just about perfectly to this graph. For researchers, who sometimes are prone to over-estimating the impact of their research results in the development of a product, this graph shows just how much effort goes into a product, beyond the basic research.

Of course, the point on the spectrum I've ignored in this discussion is depicted as the "tech transfer gap." We must ask, "How can we fill this gap?" In most organizations there is no obvious structure or management strategy to do this. Research organizations are, relatively speaking, low-budget operations, able to place only a few ~$10M bets. And product organizations strive for discipline and focus and usually can ill afford to be distracted by many small (and often competing) projects.

This is where we get back to the dogfooding culture that Rick worked hard to create. While it doesn't solve all the problems, by creating a culture where the research groups produce technologies that can be deployed throughout the organization, and where the organization broadly participates in furthering the research by "eating its own dog food," a virtuous cycle is created that helps to fill the tech transfer gap. Users create an experience base that accelerates research progress, and research progress provides a foundation for more effective experiences.

In recent years, the idea of dogfooding has waned as workers, including researchers, have adopted commercial systems for all of their computing infrastructure. But today things might be changing. Totally new infrastructure technologies are being developed, making use of cloud technologies, natural user interaction technologies, large-scale program analysis techniques, and "big data" machine intelligence and data analytics. The time seems ripe for a return to 1987, when major research organizations developed their own next-generation technologies and deployed them internally for everyday use. I am pretty sure that, at the very least, a research organization run by Rick Rashid would find this to be both productive and exciting. I look forward to it with glee.

[1] Edoardo Biagioni, Robert Harper, and Peter Lee. Signatures for a protocol stack: A systems application of Standard ML. *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming.* Orlando, June 1994.

[2] David Tarditi, Greg Morrisett, Perry Cheng, Christopher Stone, Robert Harper, and Peter Lee. The TIL Compiler for Standard ML. *ACM SIGPLAN'96 Conference on Programming Language Design and Implementation (PLDI96).* Philadelphia, May 1996.

[3] George Necula and Peter Lee. Safe kernel extensions without run-time checking. *Proceedings of the Second Symposium on Operating System Design and Implementation (OSDI96).* Seattle, October 1996, Best Paper Award winner, and winner of the 2006 SIGOPS Hall of Fame Award.

The Future of Textbooks

Rakesh Agrawal, Sreenivas Gollapudi, Anitha Kannan, Krishnaram Kenthapadi

Search Labs, Microsoft Research
Mountain View, CA, USA
{*rakesha, sreenig, ankannan, krisken*}*@microsoft.com*

Abstract

Rick, to us, is first and foremost a teacher. This article celebrates his love for education and innovation.

We postulate how textbooks will be transformed in the future, driven by their availability in electronic form and the consequent new possibilities for reading textbooks. We propose "study navigator," a tool for enhancing the reading of electronic textbooks, and describe an algorithm for constructing this tool.

1. Introduction

Education is known to be the key determinant of economic growth and prosperity [9, 21]. While the issues in devising a high-quality educational system are multi-faceted and complex, textbooks are acknowledged to be the educational input most consistently associated with gains in student learning [18]. They are the primary conduits for delivering content knowledge to the students and the teachers base their lesson plans primarily on the material given in textbooks [8].

With the emergence of abundant online content, cloud computing, and electronic reading devices such as iPad and Kindle Fire, textbooks are poised for transformative changes. Notwithstanding understandable misgivings (e.g., Gutenberg Elegies [5]), textbooks cannot escape what Walter Ong calls "the technologizing of the word" [15]. The electronic format comes naturally to the current generation of students ("digital natives" [16]) growing up surrounded with Internet, computers, video games, digital music players, mobile phones, and the other tools of the digital age. Some even assert that electronic reading devices can catalyze a new culture of global literacy [1, 6].

Electronic book technology has a long way to go before it can equal the readability and richness of traditional books [10]. Nevertheless, they have characteristics that endow them with functionality that supersedes those of traditional books [12]. Some of the novel features that can be enabled in electronic textbooks include:

- Embedded media: The book can be enriched with interactive images, picture galleries, videos, and live simulations to provide a better learning experience to the reader.
- Rich annotations: The annotations can be stored along with the book and further can be produced collaboratively by the readers of the book.
- Formative experience: Depending on the content in a section, the student can be presented with a set of questions and depending on the student's response, the content following the section can be modified. Similarly, the end-of-the-chapter exercises can be tuned to the performance of the student in the various sections in the chapter.
- Collaborative learning: The student can interact with other students while working on different parts of the lessons such as the end-of-the-chapter exercises and review questions. They can also form collaborative study groups that can span different geographical regions and time zones.
- Access: The presentation can be dynamically modified to adapt to the requirements of the reader. For example, readers with disabilities can avail themselves of bigger fonts, text to speech, and other aids.
- Updates: Corrections and minor updates to the book can be seamlessly provided to the reader.

We present the design and implementation of a study aid for electronic textbooks that we call study navigator. The study navigator can be activated by a student as she is reading a particular section. The content of the navigator for a section is algorithmically generated. Key elements of the navigator include:

- Concept references: Key concepts in the textbook necessary for understanding the ideas presented in the present section and links to the sections in which those concepts have been discussed.
- Path: The return path to the home section in case the student reached the present section following a chain of concept references.
- Augmentations: Links to external in-depth articles, images, and videos that further enhance the material presented in the section.

We restrict ourselves to discussing the first two and refer the reader to [2] for algorithms for generating augmentations.

The paper proceeds as follows. We describe the design of the study navigator as well as an

algorithm for generating the content of the navigator in §2. Finally, §3 presents conclusions and directions for future work.

2. Study Navigator

The study navigator is designed to help a student better understand the material presented in a section, exploiting the availability of the textbook in electronic form. While reading a section, the student can benefit from material pertaining to the ideas presented in the section. Such material can be obtained from two sources.

The first source corresponds to augmentations in the form of links to external in-depth articles, images and videos. References to such material can help the student obtain a deeper understanding of the concepts discussed in the section. See [2] for algorithms for generating augmentations.

The second source for such material is the textbook itself. The material could be present in other sections in the textbook and related to the ideas presented in the current section. References to such material can help the student relate the concepts discussed in the current section to other parts of the book, thereby increasing the learning capacity of the student. Having too many references to other sections in the book can cause cognitive burden to the reader. Hence we describe next an algorithm to determine a few (say k) concepts (along with underlying sections where they are explained) that would be most useful for understanding the current section (Algorithm 2.1). We also maintain the path of the (concept, section) references followed by the student, so that the student can easily return to the home section.

We first determine the set of concepts in the textbook, and the relationship between concepts. Consider a section s in the textbook. Let $\lambda_s(c,i)$ denote the significance score of concept c in a different section i for understanding section s. Given the significance scores $\lambda_s(c,i)$ of every concept in every other section for understanding section s, we then order (concept, section) pairs by their significance scores and include pointers to the top k (concept, section) pairs in the study navigator for section s. Thus the generation of the study navigator for section s requires the following inputs: (i) concepts in the textbook, (ii) relationship between concepts, and (iii) the significance score $\lambda_s(c,i)$ of concept c in section i for understanding section s. We describe next the computation of each of these inputs.

2.1 Concepts

If a textbook includes a back-of-the-book index [14], it can be used for obtaining concept phrases. Unfortunately, not all books contain such indices; e.g., in a study reported in [4], only 55% of the 113 books examined included them. Fortunately, there is rich literature on algorithmically extracting key phrases from a document that can guide the task of extracting key concepts [3, 17].

Algorithm 1 GENERATESTUDYNAVIGATOR

Input: A textbook divided into sections; A given section s; Number of desired references k.
Output: An ordered list of top k (concept, section) references for section s.

1: Determine the set of concepts C in the textbook. (§2.1)
2: Compute relationship between concepts. (§2.2)
3: **for** each section i (different from s) and each concept c occurring in section i **do**
4: Compute significance score $\lambda_s(c, i)$ of concept c in section i for understanding section s. (§2.3-§2.5)
5: Return an ordered list of top k (concept, section) references for section s, based on the significance scores.

After studying several textbooks, we devised the following approach. Following [2, 11], we define concept phrases to be terminological noun phrases. We first form a candidate set of phrases using linguistic patterns, with the help of a part-of-speech tagger [17]. We adopt the pattern A^*N^+, where A refers to an adjective and N a noun, which was found to be particularly effective in identifying concept phrases. Examples of phrases satisfying this pattern include 'cumulative distribution function', 'fiscal policy', and 'electromagnetic radiation'. The initial set of phrases is further refined by exploiting complementary signals from different sources. First, WordNet [7], a lexical database is used to correct errors made by the part-of-speech tagger. Next, both malformed phrases and very common phrases are eliminated, based on the probabilities of occurrences of these phrases on the Web, obtained using Microsoft Web N-gram Service [19]. The reason for eliminating common phrases is that they would be already well understood.

2.2 Relationship between Concepts

We first attempted to induce relationships between concepts by mapping concept phrases to Wikipedia articles and use the link structure between the Wikipedia articles to infer relationship between concepts. We discovered the following issues. Many Wikipedia articles have asymmetric hyperlink structure, plausibly due to the encyclopedic nature of Wikipedia: there are relatively less links from articles on specialized topics to articles on more general topics. For instance, the Wikipedia article titled 'Gaussian surface' mentions 'electric field' 11 times but does not have a link to the latter. Furthermore, while Wikipedia provides good coverage for universal subjects like Physics and Mathematics, it has inadequate coverage for concepts related to locale-dependent subjects such as history.

We, therefore, derive the relationship between concepts directly from textbooks using co-occurrence. More precisely, we defined $R(c)$ to be the set of concepts (including c) that co-occur with c in at least e sections such that both c and the co-occurring concept c' occur within a window of size l in each of these e sections. The requirements of co-occurrence

in multiple sections and co-occurrence within a window size ensure that we only consider concept pairs that are significantly related to each other.

2.3 Significance Score

The significance score $\lambda_s(c,i)$ is a measure of how significant is the description of concept c in section i for understanding concepts in section s in the book. A naive approach would be to define the significance score in terms of the relative frequency of the concept phrase in the section, for example, $\lambda_s(c,i) := (freq(c,i)) / (\sum_{1 \le i \le n} \sum_{c \in C} freq(c,i))$. A problem with such a definition is that it does not differentiate between two concept phrases c_1 and c_2 with the same frequency in a given section. However, concept c_1 may be related to many concepts that occur in other sections in the book and hence more significant for understanding an arbitrary section in the book, while c_2 may be relevant only for the current section. A work-around might be to take into account the number of related concept phrases of each concept, for example, $\lambda_s(c,i) \propto freq(c,i) \cdot |R(c)|$. However this definition suffers from the fact that even two concept phrases with same frequency in a section and same number of related concept phrases may have very different significance for understanding a different section s, depending on how connected are the related concepts themselves to other concepts in the book and concepts in section s in particular, and how often the concept and its related concepts occur in other sections.

To address these issues, we define the significance score of a concept phrase in section i for understanding a different section s taking into account the following factors: (a) how frequent is the concept in section i, (b) how many concepts are related to it, and (c) how likely is the description of this concept in section i to be referred when a reader is trying to understand a concept in section s. How do we formalize and quantify (c)? We surmise that while reading a book, a reader would refer to more significant concepts more often. Thus, we determine how likely is the reader to refer to concept c in section i when reading about an arbitrary concept in a different book section and then aggregate over each concept in section s of the book to obtain the significance score $\lambda_s(c,i)$.

2.4 Reader Model

Consider a reader who reads a textbook sequentially starting from the first section. When she is reading a section i, she comes across the concept phrases in the order $c_{i1}, c_{i2}, c_{i3}, \ldots$. When the reader comes across a concept phrase, with a large probability, the reader will be persistent in continuing to read the section. With a certain probability, she may not understand the concept and hence may be forced to refer to another section to seek explanation.

Postulate that whenever the reader does not understand the concept phrase c, she refers to a section containing the same concept phrase c or a concept phrase related to c. More precisely, she picks a related concept c' from $R(c)$ with equal probability, chooses an

occurrence of c' amongst all occurrences of c' in the book with equal probability, and refers to the corresponding section i' to learn more about c'. After reading about c' in i', the reader has the following options: (a) return to the original section i with a large probability, and continue further reading, or (b) digress further to learn more about c' by referring to a section containing c' or a concept related to c', that is, pick a related concept c'' from $R(c')$ with equal probability and refer to a section i'' that contains c'' amongst all occurrences of c'' with equal probability. In the latter case, the reader then returns to the original section i, or digresses further.

2.4.1 Random Walk over Concept Graph

We formulate the above process as a random walk [13] over a concept graph $G = (V, E_p \cup E_d)$. Each node $u = \langle i, c_{ij}, j \rangle \in V$ is a $\langle$ section, concept, position $\rangle$ triplet corresponding to the occurrence of concept phrase c_{ij} in section i and its sequential position j amongst the concept phrases in the section. Denote the associated section i by $\overline{i}(u)$ and the associated concept c_{ij} by $\overline{c}(u)$. There are two types of directed edges in G. The set of persistence edges E_p consists of directed edges corresponding to sequential reading of the book, that is, there is a directed edge from $\langle i, c_{ij}, j \rangle$ to $\langle i, c_{i(j+1)}, j+1 \rangle$ and from the last concept node in a section to the first concept node in the next section. The set of digression edges E_d consists of directed edges corresponding to forced digression, that is, there is an edge from u to v if $\overline{c}(v) \in R(\overline{c}(u))$.

The random walk consists of three types of transitions:

1. Persistence transition: From any node u, follow the persistence edge, that is, the reader persists to read sequentially from the concept occurrence corresponding to u. Denote the probability associated with such a transition as the persistence factor, α.

2. Digression transition: From any node u, follow a digression edge. Denote the total probability associated with a transition along one of the digression edges outgoing from a node as the digression factor, β. Suppose the reader picks a related concept $c' \in R(\overline{c}(u))$. The reader then selects an occurrence of c' amongst all occurrences with equal probability.

3. Diligence transition: From any node to which the reader has digressed, return to the node from where the digression originated. This transition corresponds to the reader returning back to the starting point after a digression. Denote the probability associated with such a transition as the diligence factor, γ.

2.4.2 Relationship between α, β, and γ

We next show that there is effectively one parameter. When digression originates from a node, there are exactly two choices, to persist reading or to digress, and hence $\alpha + \beta = 1$. Similarly, for subsequent nodes in the digression, there are exactly two choices, to return back to starting node or to digress further, and hence $\gamma + \beta = 1$. Thus $\alpha = \gamma = 1 - \beta$. This relationship between α and γ is in agreement with the following natural intuition: one's tendency to read forward in a section is the same as the tendency to return to the starting point after a digression, since both these tendencies try to achieve the same goal of one's disciplined reading and completion of the entire book.

2.5 Computing Significance Scores

Consider the random digression walk starting from an arbitrary node u (that is, the walk corresponding to the chain of digressions originating from u consisting of only digression and diligence transitions). This walk induces a Markov chain over the strongly connected component reachable from u and further this Markov chain is ergodic. The claim below follows from the fundamental theorem of Markov chains [13].

> *Claim 2.1 There is a unique stationary probability distribution $\pi(u,.)$ associated with the random digression walk starting from any node u in G.*

By definition, the stationary probability $\pi(u,v)$ denotes the probability that the walk starting from node u is at node v in the steady state. In other words, this probability corresponds to the relative frequency with which the reader refers the concept $\overline{c}(v)$ corresponding to v when trying to understand the concept corresponding to u and hence larger $\pi(u,v)$ implies that the reader is more likely to refer to v. Thus $\pi(u,v)$ is a measure of the relative significance of an occurrence of concept $\overline{c}(v)$ in section $\overline{i}(v)$ corresponding to v for understanding the concept corresponding to u. Considering the random walks starting from each concept node in a given section s of the book, we can thus compute the significance of a single occurrence of concept $\overline{c}(v)$ in section $\overline{i}(v)$ for understanding concepts in section s. Our goal is to compute the significance of all occurrences of a concept in a section. Hence we further aggregate the above score over all occurrences of concept $\overline{c}(v)$ in section $\overline{i}(v)$. In this manner, we also incorporate the frequency of the concept in the section.

We thus define the significance score $\lambda_s(c,i)$ of a concept c in section i for understanding section s in terms of the combined stationary probability associated with nodes corresponding to all occurrences of c in i, summed over random walks starting from all concept nodes in section s. We remark that our definition of $\lambda_s(c,i)$ takes into account the desired factors discussed in § 2.3: the frequency of c in i, the number of concepts related to c and the

likelihood that the description of c in i would be referred for understanding concepts in section s in the book.

Definition 2.2 Given the stationary probabilities $\pi(.,.)$ associated with the random digression walks, define the significance score of a concept c in section i for understanding section s as:

$$\lambda_s(c,i) = \sum_{v \in V : \overline{i}(v)=i, \overline{c}(v)=c} \quad \sum_{u \in V : \overline{i}(u)=s} \pi(u,v).$$

2.6 Computing Significance Scores

Consider the random digression walk starting from an arbitrary node u (that is, the corresponding walk induces a Markov chain over the strongly-connected component reachable from u and further this Markov chain is ergodic. The claim below follows from the fundamental theorem of Markov chains [13].

Claim 2.3 There is a unique stationary probability distribution $\pi(u,.)$ associated with the random digression walk starting from any node u in G.

By definition, the stationary probability $\pi(u,v)$ denotes the probability that the walk starting from node u is at node v in the steady state. In other words, this probability corresponds to the relative frequency with which the reader refers the concept $\overline{c}(v)$ corresponding to v when trying to understand the concept corresponding to u and hence larger $\pi(u,v)$ implies that the reader is more likely to refer to v. Thus $\pi(u,v)$ is a measure of the relative significance of an occurrence of concept $\overline{c}(v)$ in section $\overline{i}(v)$ corresponding to v for understanding the concept corresponding to u. Considering the random walks starting from each concept node in every other section of the book, we can thus compute the significance of a single occurrence of concept $\overline{c}(v)$ in section $\overline{i}(v)$ for understanding concepts in other book sections. Our goal is to compute the significance of all occurrences of a concept in a section. Hence we further aggregate the above score over all occurrences of concept $\overline{c}(v)$ in section $\overline{i}(v)$. In this manner, we also incorporate the frequency of the concept in the section.

We thus define the significance score $\lambda(c,i)$ of a concept c in section i in terms of the combined stationary probability associated with nodes corresponding to all occurrences of c in i, summed over random walks starting from concept nodes in all other sections. We remark that our definition of $\lambda(c,i)$ takes into account the desired factors discussed in § 2.3: the frequency of c in i, the number of concepts related to c and the likelihood that the description of c in i would be referred for understanding other concepts in the book.

Definition 2.4 Given the stationary probabilities $\pi(.,.)$ associated with the random digression walks , define the significance score of a concept c in section i as:

$$\lambda(c,i) = \sum_{v \in V : \bar{i}(v)=i, \bar{c}(v)=c} \quad \sum_{u \in V : \bar{i}(u)=i} \pi(u,v).$$

We remark that multiple alternatives exist for computing the above inputs and our model of study navigator can admit multiple such choices. For example, the significance score computation can benefit by including anaphoric references to a concept when computing the occurrences of concept phrases in a section.

3. Conclusions and Future Directions

The future of textbooks is electronic. Sven Birkerts thus opined about this brave new world [5]: "What the writer writes, how he writes and gets edited, printed and sold, and then read—all the old assumptions are under siege." However, the current technology is still quite nascent. We anticipate a surge of innovations to make learning from electronic textbooks much more pleasant and productive. We contributed a step in this direction by presenting the design of a study navigator that can help a student learn the material better and faster.

For the future, we would like to apply the ideas presented to some textbooks and carry out experiments to study their effectiveness. We would also like to explore the design of other tools to help electronic textbooks fulfill their potential.

4. References

[1] Anda Adams and Jacques van der Gaag. First Step to Literacy: Getting Books in the Hands of Children. The Brookings Institution, 2011.

[2] Rakesh Agrawal and Sreenivas Gollapudi and Anitha Kannan and Krishnaram Kenthapadi. Data Mining for Improving Textbooks. SIGKDD Explorations, 2012.

[3] Anderson, J.D. and Pérez-Carballo, J. The nature of indexing: how humans and machines analyze messages and texts for retrieval. Part II: Machine indexing, and the allocation of human versus machine effort. Information Processing & Management, 37(2), 2001.

[4] Bakewell, K.G.B. Research in indexing: more needed?. Indexer, 18(3), 1993.

[5] Sven Birkerts. The Gutenberg Elegies: The Fate of Reading in an Electronic Age. Faber & Faber, 2006.

[6] Suzie Boss. What's next: Curling up with e-readers. Stanford Social Innovation Review, 2011.

[7] Christiane Fellbaum. WordNet: An electronic lexical database. MIT Press, 1998.

[8] Gillies, J.A. and Quijada, J.J. Opportunity to Learn: A high impact strategy for improving educational outcomes in developing countries. USAID Educational Quality Improvement Program (EQUIP2), 2008.

[9] Eric A. Hanushek and Ludger Woessmann. The Role of Education Quality for Economic Growth. Policy Research Department Working Paper 4122, World Bank, 2007.

[10] Terje Hillesund. Will E-books change the world?. First Monday, 6(10), 2001.

[11] J. S. Justeson and S. M. Katz. Technical terminology: Some linguistic properties and an algorithm for indentification in text. Natural Language Engineering, 1(1), 1995.

[12] Steven Levy. The Future of Reading. Newsweek, 2007.

[13] Motwani, R. and Raghavan, P. Randomized Algorithms. Cambridge University Press, 1995.

[14] Mulvany, N.C. Indexing books. University of Chicago Press, 2005.

[15] Walter J. Ong. Orality & Literacy: The Technologizing of the Word. Methuen, 1982.

[16] Marc Prensky. Digital Natives, Digital Immigrants. On the Horizon, 9(5), 2001.

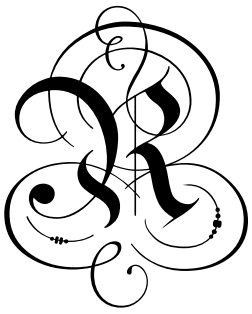
[17] S. Sarawagi. Information extraction. Foundations and Trends in Databases, 1(3):261–377, 2008.

[18] Toutanova, Kristina and Klein, Dan and Manning, Christopher D. and Singer, Yoram. Feature-rich part-of-speech tagging with a cyclic dependency network. NAACL-HLT, 2003.

[19] A. Verspoor and K. B. Wu. Textbooks and educational development. Technical report, World Bank, 1990.

[20] Kuansan Wang and Christopher Thrasher and Evelyne Viegas and Xiaolong Li and Paul Hsu. An Overview of Microsoft Web N-gram Corpus and Applications. NAACL-HLT, 2010.

[21] World-Bank. Knowledge for Development: World Development Report: 1998/99. Oxford University Press, 1999.



ick Rashid, Renaissance Man

And Now, the Next 20 Years

Gordon Bell with Jim Gemmell

In May of 1991 Nathan Myhrvold asked me to help plan the yet-to-exist Microsoft Research. "I will help you find someone really good to lead it," I promised. On this 21st anniversary of MSR and your 60th birthday, I am delighted that you have not only lived up to my expectations, but exceeded them!

Nathan and I had half a dozen criteria besides just leading teams to promised lands and walking on water. Notably, we demanded proof that whoever was to head MSR had actually built something worthwhile that worked, was in use, and being built on. You overachieved on this one with Mach, and thus occupied the top of our list to head Microsoft Research ahead of scores of your peers. We just needed to validate that you had catholic tastes in computing, despite spending a large fraction of your youth building a Unix dialect. We went to talk with you on the morning of June 25th and were ecstatic to find that you and your family actually owned and operated Windows computers.

We were sold. I delegated the recruiting to Nathan and Bill Gates, who wined and dined you in Redmond and made you a formal offer.

You refused.

Nathan called me in early August with the news. After reprimanding him and Bill for blowing it, I spent the weekend writing you and your family a long letter on Why You Can't Turn Down The Offer. It was probably my most influential, yet un-cited work!

The letter was pretty simple: here's how you impact computing; you can't find a greater challenge than starting a new research lab with a chance to make it the world's best; the pay is OK; Bill and Nathan are offering essentially unlimited support, freedom, and budget … and if you leave CMU you will be free of the Mach code base you were supporting. And then I added:

> *"Here, I see Microsoft as the best place to lead and carry out research because you can transfer results to product so easily. Furthermore, the budget is not going to be a constraint. Being best at technology transfer will be one of the lab's greatest contributions."*

Calling product transfer easy might sound like a recruiter's glib lie, but I was sincerely swept away by the excitement of MSR's momentous beginning, and ended up making this unusually naïve and optimistic statement.

Technology Transfer Isn't so Easy

Technology transfer is never easy. Microsoft's product engineers may be the most difficult in the world to transfer technology to because they are so competent and demand that the transferee show deference, yet have superior ideas, skill and knowledge. Unless they are being brutally assaulted in the marketplace to the point of desperation, their patience and motivation to accept ideas from others is close to nil.

A product project is like a train and the objective is to somehow put something on the train that makes it better or maybe even changes its destination. If your timing isn't just right as it leaves the station you will miss it—or get run over by it. Sometimes, if you want to be part of the trip, you have to join the crew. In a few very special instances you can create a new train with a new destination.

You should be exceptionally proud that "every Microsoft product contains a contribution from research." Some contributions are as small as a paperclip; at the other extreme is the immense impact of Kinect—the voice and computer vision technologies to create a new user interface that *actually works*. The contribution of this gesture-speech interface may be akin to that of the legendary PARC UI of the 70s. We'll know in a decade and the measure will be the size and scope of platforms it enabled. We'll know when "hand waving" actually makes something we want, happen.

Looking Back to Look Forward

I posited the Law of the Birth and Death of Computer Classes that I believe explains the computer's 60-plus-year evolution (Bell, 2008). This corollary to Moore's Law says that approximately every decade a new computer class comes into existence creating a new industry segment in a new price range and *a distinctively new platform enabling new uses and new companies*. The platform class usually means a new interface to people or other information processors and maybe a new supporting network. Classes are enabled by shifts in hardware cost with a new software approach and platform. NUIs are just an enabler for the new classes of smart phones and tablets, allowing us to use our fat fingers to read things. I call them little reader-computers.

In 1991 the World Wide Web platform arrived and by 2000 it had encompassed smart phones as clients. While Microsoft was instrumental in creating the first tablets, by 2000 small media

players and tablets were born as reader-computers without us. By 2005, it was clear that the massive number of users and devices would require unlimited central computers—thus, "cloud computing" represented in a new, back to the past computer at a millions of processor scale unlike anything I had been able to foresee. Consequently, the last twenty year timeline is the most dense in terms of size and computer class diversity—all the facets of the web from consumption to communication, media players and readers, communicators, etc.

Robots of all sorts are walking, swimming, and flying into our lives, including those that are just there being communication channels, even chatting and trying to help. The next twenty years will be even more dense in computer classes because of what the cloud affords: unlimited computing, storage, and universal network access at low cost and NO capital outlay if you are starting a company. More companies and businesses than we ever dreamed of will come into existence and they will be about more than just "spilling our guts" to whomever to be "social"—your next challenge is to see that one or two of them will come from Microsoft Research.

The Next 20 years

Much of the time, the law of computer classes has been about computers doing pretty much the same old jobs, a couple of decades later and at mass consumer prices with a different UI. We saw this when 1980s PCs and workstations did pretty much the same job as terminals connected to 1960s mainframes and 1970s minicomputers. By the 1990s, WIMP enabled more users through pointing versus learning arcane syntax. Then 10 years later, the reader-computer in smart phone and book/reader form factors connected to the cloud have taken over the lion's share of the PC's "face time." (In 2005 and then again in 2007, I recall trying to bet the MSR Technical Advisory Board on the day that this cross-over would happen—nobody would take my bet since these computers were far from our research, hearts, and minds then).

The cloud is now a well-established platform that is indeed the ubiquitous utility that J.C.R. Licklider and others described 50 years ago. It is the basis on which things should get really interesting in the next two decades. The cloud can support the connections to everything digitizable, store the conversations, and be able to process and relate this information to all the other connections. Within a decade, there's no reason to expect anything less than a trillion[1] personal information sensing and effecting devices that will be connected through it. For example, an embedded system exists that fits in a one-millimeter cube; it could become a general-purpose platform inside our bodies. Dust-sized sensor services are being deployed that can gather and monitor physiological signals through a brain-computer interface (BCI).

With cloud computing, Nathan's Law rules—"software is like a gas that fills every container it is in, and the only constraint is our imagination." In the case of the cloud, costs trend to zero and the size is unlimited. Virtually everything we build will be connected to the cloud in some way, which means it is a technology that needs to be an essential part of computer science training. Maybe we can help bring this about.

So I see a number of important programmable platforms that will emerge based on unlimited,

inexpensive computation, lifetime storage, and-all-the-time, everywhere network access. Let's consider some of the platforms destined to emerge in the next 20 years.

IOT: The Internet of Things; and the Internet of Other Stuff

I'm not always sure what people mean by the term, but I think the Internet Of Things (IOT) is always mentioned because we can observe the emergence of dust-sized computers. Bing knows there are almost 2 million places on the web that talk about them. Wikipedia says "uniquely identifiable objects (things) and their virtual representations in an Internet-like structure." The Wikipedians go on to state the obvious:

> *"With all objects in the world equipped with minuscule identifying devices, daily life on Earth would undergo a transformation. Companies would not run out of stock or waste products, as involved parties would know which products are required and consumed. Mislaid and stolen items would be easily tracked and located, as would the people who use them. Your ability to interact with objects could be altered remotely based on your current status and existing user agreements."*

A decade ago, I posited a similar scenario. Every object would be given digital identity, memory, and ability to communicate at birth. Imagine every piece of sheetrock having this digital life from its manufacture in a factory through its transport, sale, and eventual residence in a home or office. In order to do this and other more near-term things, a really robust and user-friendly home network is essential.

In addition to the obvious devices that require sensors and effectors, everything will have an identity and digital life. I wonder where this life history is stored and maintained? Is it embedded in an artwork or jewelry or clothing, or are these items just identified somehow with a key, and their lives are held in the cloud. The Digital Life of Stuff will emerge slowly, but the need to keep track of everything is inevitable and no more than a decade away from emerging.

Robots as Controllable, Telepresent Communicators, Transporters, and Useful Mates

Within a decade some research team or new startup company will create a serious Anthropomorphic programmable robot platform. I can't tell you who will do it, or describe exactly what it will do. It will be incredible and the sad thing for me is only being able to glimpse it as an infant. You, on the other hand, will see it maybe even as an adolescent. When MSR started, I suggested telepresent robots as a research area; now there are several of these that the idea spawned, including InTouch Health. Making such a computer and platform that people work on by filling its brain with software is unimaginable. Will we play here? Or are we limited until computer science departments produce enough people who are working on this to spawn the new companies?

On- and In-Body Platforms for Health, Wellness, and a Better, Bionic Being

These exist now and the rate of new device introduction is probably the highest of all time. Many are smart phone peripherals since "a smart phone is an on-body, personal mainframe." I'm on my second in-body computer that controls my heart. Last month, I got a new lease on life when we discovered that it was set improperly and was driving me to an earlier death—something I diagnosed by using an external monitor. This should have been done by the device automatically—even doctors and technicians are often challenged by these platforms. Analyzing the bits that come from these devices is a wholly new business because now they are *all* ignored, yet are the key even to predicting time of death.

These embedded devices are wonderful, but they need to be monitored, understood, and adjusted automatically, rather than being subject to people setting modes and parameters and occasionally musing over their output. For me, much better in-body monitoring is essential. The BCI will be a big part of this story as well. For convenience, I'd also be happy with a fool-proof bionic or electronically-implanted identity as well as a septic RJ45.

MyLifeBits: Extreme lifelogging, personal information system, and digital immortality

Your support of our work on lifelogging for personal archiving continues to expand and is gradually encompassing the personal archiving community ranging from personal, to academia and museums, and on to the Library of Congress and British Library. National libraries are responsible for long-term preservation of individual archives. At the root is: how do you encode a life and save it forever? Who, how much of their life, and for what purposes? Equally important is the notion of extreme lifelogging whereby everything in a person's life is captured. And there's a great challenge when all these past life bits are called on to be able to serve a role for personal immortality. Indeed, we believe MyLifeBits is really the trajectory of personal computing progress. A list of *New York Times* predictions sees "Full Life Recording" within 10-20 years.

With our MyLifeBits project, just describing it (Bell, G. and J.Gemmell, 2009; 2010), has evoked dozens of projects, startup companies and platforms[2]. Lifelogging enables the use of computers as the ultimate, long-term and forever memory. We believe that Extreme Lifelogging that captures "everything" i.e. everything seen and heard, will be commonplace in a decade. For example, the team at Evernote claims to have 20 million users and is concerned with living to be 100—they also claim that MyLifeBits was the inspiration for the company. The Timeline Feature of Facebook is a wonderful way to view a person's lifelog. So mostly, our "technology transfers" have been to new companies. We hope that these products will be as useful as we envisioned, creating new markets and businesses, to which our product groups will respond when the size is right.

Conclusion

The next 20 years will bring us new computer classes and UIs in rapid succession. I dream

about—and hope I will get to see—robots, the Internet of things, body platforms and lifelogging that can be the basis of immortality for our progeny. Like so much of computing progress, these will come about based on enlightened self-interest as we build things to satisfy ourselves.

This time I won't sugar-coat it: technology transfer is hard. But I remain optimistic. As hard as it will be to get on these trains before they leave the station, I know where to look to have my expectations exceeded. Good luck on this quest.

Acknowledgement

Thanks to Jim Gemmell, my colleague and collaborator, for valuable content and delivery assistance.

Bibliography

Bell, C. G. (2008, January). Bell's Law for the birth and death of computer classes. Communications of the ACM, 86-94.

Bell, G. and J.Gemmell. (2009;2010). Total Recall; and Your Life, Uploaded (Paperback). New York: Dutton; Penquin.

Endnotes

1 A trillion may be a low estimate. It would be 140 per person or if covering the 150 million square kilometers of earth, it means one per 1500 square meter. On streets and roadways, one every few 10s of meters.

2 Quindi, WisdomArk/MemoryArk, many Digital Cemeteries, reQall, Lifebio, IBM Penseive, Famento, MyCyberTwin, deepvue, egoArchive, Memolane, Erly, Calltrunk, Dadapp, socialsafe, lazimeter, memdedn, Evertale, Timehop, etc.

Rick Rashid—Aeroplane Pilot

Andrew Herbert

23 February 2012
Cambridge, UK

One of the pleasures of being a Microsoft Research Lab Director is hosting an annual showcase event attended by Rick, the directors of the other labs and colleagues from across MSR. For the Cambridge Lab, I made the focus our Technical Advisory Board and generally tried to arrange a novel venue for the TAB dinner. In 2009 we visited Bletchley Park and saw the replica of Colossus, the world's first electronic computer, built in 1943 to break the German "Tunny" teleprinter codes. In 2010 we visited the Imperial War Museum at Duxford to see their collection of historic military planes and as special treat I asked a good friend, Clive Denney, who is a qualified air display pilot to give a short show using the 1953 de Havilland DHC-1 Chipmunk military training aircraft I had acquired earlier that year. Having taken Rick and his family on an extended visit to Duxford on an earlier trip, I knew he was interested in vintage aviation and that his father had flown from East Anglia with the American Eighth Air Force during the Second World War so I asked Clive to take Rick for a quick tour of the area around Duxford in the Chipmunk while the rest of us toured the museum. Returning in time for the dinner it was clear from the huge grin on his face that Rick had enjoyed the experience and perhaps that was where the seed was planted that he would like to get his own pilot's licence. Now two years later, at the time of writing, Rick is well advanced in his training, having passed the stage of "first solo" and early solo navigation sorties and on the threshold of taking his qualifying check ride. The accompanying photograph shows Rick walking out to meet Clive in front of the Chipmunk.

As it happens, both trips (Bletchley and Duxford) planted seeds for me: having chosen to "retire," I now have two projects underway. The first is to build a replica of the first Cambridge computer for Bletchley Park. The machine was called EDSAC and ran it's first program on 6th May 1949. EDSAC is important as it was the first computer to provide a computing service and the first machine to have a symbolic assembler and loader for the input of programs. The EDSAC replica will take the Bletchley Park story on from the wartime Colossus to the restored

1950s and 1960s machines in the museum. My second project is restoring a 1937 vintage Royal Air Force Walrus floatplane to flying condition. The Walrus is an important part of British military aircraft history for its significant role in rescuing shot-down Battle of Britain pilots from the English Channel in 1940 and its use as a submarine spotter defending British convoys at sea.

Knowing Rick was in charge of Microsoft Research was one of the key reasons I was comfortable accepting Roger Needham's invitation to join the Cambridge Lab in 2001. I believe we had first met at the 1979 Symposium on Operating Systems Principles at Asilomar: Rick was presenting his work on the "Rochester Intelligent Gateway" and I was part of the Cambridge team reporting on the Cambridge CAP Capability Computer. Our paths continued to cross over the following decades through our mutual research interest in operating systems and distributed computing.

I owe a great debt to Rick for making me Managing Director of the Cambridge Lab following the sad loss of Roger Needham to cancer in 2003. He was tremendously supportive in allowing me to grow the lab from the 50 people Roger had collected to the 150 or so it has become today. I had an amazing time adding two new groups to the organization (Stephen Emmott's Computational Science group and Ken Woodberry's Computer Mediated Living group) and hiring many talented people. With Rick's encouragement I connected the Cambridge Lab more closely with the EMEA academic community, found new routes to technology transfer "at a distance," and ways of supporting the Microsoft organization in our region. Rick has a unique way of challenging and guiding those who work for him that allows them to develop both themselves and their organizations with great success. He gives his lab directors a great deal of autonomy, supports their new ventures, gives credit and values success and, on those occasions when things don't work out as planned, constructively helps resolve problems and issues. I found the environment he provided to be highly stimulating and rewarding.

In summary I would like to acknowledge Rick for his vision and leadership in creating and sustaining Microsoft Research as a world-leading computer science laboratory, for the trust he showed in me and the Cambridge Lab, and the support he gave us. I wish him "happy landings" as a pilot and finish with an open invitation for us to spend some time together on one of his future UK trips developing his "stick and rudder" skills in my Chipmunk.

Randomness, Science, and the Library of Babel

Greg Bear

It's been tremendous fun and a real privilege over these many years to hang out with Rick and Terri and all the creative folks at Microsoft Research. Our discussions over dinner or conference tables (or conference tables and then dinner, or box lunch) have always been lively and informative, and have helped keep me on a relatively straighter and narrower path than I might have otherwise taken...

But in celebration of where a few twisty meanders can take us, I offer up this piece, based in part on researches conducted while writing DARWIN'S RADIO and CITY AT THE END OF TIME.

I'm sure Rick and Terri remember many of *those* discussions!

Imagine a library filled with every possible book. Each book is, say, 410 pages long. Each page contains forty lines, and each line, eighty characters, chosen from a total alphabet of twenty-five letters. The library consists of all possible permutations of those letters in each volume. That is, volumes may differ by just one letter, two letters, by a single word, or by whole passages and pages. Most volumes will differ completely from their fellows.

How long would it take to A: build all those shelves? B: fill all those shelves with books? C: dust all those books?

Half-seriously, writers and philosophers have been thinking about such a library for centuries. Perhaps the most famous example (the configuration described above) is found in "The Library of Babel," a speculative essay by Jorge Luis Borges. I first encountered the idea either in George Gamow's *One, Two, Three, Infinity* (first published in 1947) or in Kurd Lasswitz's 1901 story, "The Universal Library," reprinted in *Fantasia Mathematica,* edited by Clifton Fadiman (1958).

Recent speculations about a "universal library" of all human knowledge, digitized and available over the Internet, are certainly ambitious, but pale in comparison to this larger effort.

How big would this hypothetical universal library be? On *Wikipedia,* we are told that Borges's library would consist of a staggering number of volumes:

$25^{1,312,000}$, or $1.956 \times 10^{1,834,097}$

One trillion is 10^{12}.

Even variations on a single paragraph—with all the rest of the text remaining unchanged—would fill a space larger than our universe. Such enormous numbers may limit our acceptance of the very idea. It seems ridiculous to imagine so many texts, or in more cybernetic nomenclature, so many *strings*. This type of string is defined in the OED as a "linear sequence of characters, words, or other data."

Yet we are fascinated by infinities, and there is a great deal of mathematics that deftly describes, ranks, and even operates with infinities, though of course any infinite number goes on forever. The above huge number of volumes could fit into an infinite number an infinite number of times.

Why, then, should we be daunted by very, very, very *large* numbers?

In fact, science and mathematics are filled with ideas which involve extraordinarily large numbers. In the eighteenth century, Gottfried Leibniz speculated that if we could only know the exact position of every particle in the universe, we could predict the next state—the universe's immediate future. This idea codified a deep assumption beneath all notions of a "rational" clockwork universe. Of course, knowing these particle positions requires not just observing and measuring, but storing that knowledge, and then running calculations to determine the next position of each particle, and then calculating how they will all interact... A pretty idea that soon gets completely out of hand. So challenged, we might then blush and claim, blithely enough, that in the future, a calculating engine of sufficient capacity and speed might be able to do just that... Which takes us out of the realm of philosophy, and into the realm of science fiction!

Our modern quantum world apparently precludes such completely rational predictions. The position and momentum of a particle cannot both be known at once, we are told—and experiment so far confirms Heisenberg's famous uncertainty principle. This indeterminacy has been explained or described in a number of ways. "Many worlds" interpretations say that for each changing state of a particle, there is a very large, perhaps infinite number of branching states that lead to a very large, perhaps infinite number of alternate worlds. This turns the universe, or *multiverse*, into a vast collection of world-lines, each one of which branches again and again, ad infinitum. *To infinity.* This is a far more daunting idea than the Library of Babel.

More intriguing to me is the prospect of multiply branching universes with differing physical laws and constants. These seeming infinities may collapse into finitude along predictable principles: those which are the most logical, the simplest, or require the "least energy." (Or perhaps which are most easily calculated—which would be convenient!)

Others have reasoned that only those universes exist that can generate observers such as ourselves. (The "Anthropic Principle.") All those other universes or branches are "virtual," never quite real, and either quickly vanish into the quantum foam, or simply don't count.

There are other kinds of universal libraries. One can imagine a "picture library" of all possible variations of a 1000x1000 pixel image, each pixel having a range of, say, twenty-five colors. Strangely, this picture library would have as a subset the universal library of texts described above—since it would contain reasonably high-resolution images of every single page in every single book. (But not all the combinations of those pages you would find bound up as individual volumes in the Library of Babel.)

And conversely, the universal library of texts would contain word and coordinate descriptions of every single 1000x1000x25 color image, embedded in many different languages and codes!

One quickly realizes that though these libraries are vast, no single volume or picture or collection can be easily dismissed as nonsensical or empty of interest. Seemingly random arrangements of letters or pixels might contain hidden meanings—for example, the coordinates of an image, or hidden references to other resources within the library, such as the perfect card catalog. And buried within "readable" texts might be additional nested codes and clues, which could give even Martin Gardner's celebrated Dr. Matrix a very long lifetime of curious pleasure.

Many strings would doubtless refer to non-alphabetic texts by using combinations of symbols to denote individual hieroglyphs or ideograms, much as modern-day computers handle Chinese and other languages. Within an alphabetic universal library, then, lies a complete library made up of any finite set of ideograms, hieroglyphs, or combinations thereof.

When discussing ideas about universal libraries with my friend, physicist Gregory Benford, he sniffed and said, "But that's just *noise*." I was taken aback for a moment, almost convinced he was correct. But within a few minutes, I had an answer to his objection: *noise* is allowed to repeat itself. (Whether it does or not is another interesting discussion.) But our library can't repeat itself. While portions of the selected strings—sentences or pages long—may be the same, no volume will be an exact copy of another volume. At least one symbol in the string—one letter in the longer text—must differ uniquely. And weeding out such repetitions, or not creating them in the first place, is hardly trivial!

Moreover, there are quite real strings which the universal library *will not* contain. Among these is a complete readout of *pi*, the ratio between the diameter of a circle and its circumference, roughly 3.1415... etc. The "tail" of *pi*, past the decimal point, goes on forever, and appears to be random. Some have called this kind of number (along with the square root of 2, another

example) "pseudo-random," because they can be generated by algorithms much smaller than they are. And that leads us into one definition of randomness, devised by Chaitin[1], Kolmogorov, and Solomonoff: mathematically speaking, a random string is any string that cannot be generated by any other string shorter than itself. Algorithms are frequently used to generate pseudorandom numbers; if you don't know the "trick," so to speak, you won't be able to predict the sequence of the number. But knowing the trick, you can reproduce the number exactly.

When Carl Sagan, in his novel *Contact,* supposed that *pi* might contain a hidden message, many mathematicians solemnly tut-tutted. A random number *cannot* contain a linguistic message, they claimed. Yet *pi* is itself is pseudo-random, and has never formally been proven to be random. Since *pi* is infinitely long, an interesting question arises—how many universal libraries, coded as numbers, will any sufficiently lengthy string-segment of *pi* contain? Or does the primary, algorithmic nature of *pi* preclude *any* meaningful messages contained therein?

Is *pi* its own and sole message?

How far down the sequence of random numbers in *pi* must we travel to "leave the territory" of the original algorithm, and wander into a generally haphazard realm that could possibly be found in other infinitely long, random or pseudo-random numbers? Interesting questions which might or might not lead us to a solution for Carl's conundrum.

Our universal library will of course contain equations and descriptions sufficient to reconstruct *pi*—and perhaps even proofs of its randomness. (But be careful: most of those proofs will be false.)

Thinking about *pi* leads us to one possible way to begin exploring a universal library. *Pi* apparently refuses to be "zipped" or compressed. Pseudo-random strings refuse compression except for the algorithm or algorithms that generate them in the first place. "Noisy" pictures on a digital camera take up more memory than less noisy images because they resist compression, and because so far, cameras are not endowed with the necessary expertise to recognize all of the pixels which are random bits of noise, and those which are significant parts of the picture—such as a star in a star field. Even the most sophisticated noise reduction algorithms in software typically sacrifice real information along with the noise.

Truly random strings all have this property: they cannot be digitally squeezed into smaller versions and reassembled with complete fidelity, as happens when we compress, say, this essay into a zip file on a hard drive.

It seems logical to assume that all the "useless" or meaningless volumes within a universal library will likewise resist compression. Whole spaces of our library could thus be explored, then processed or "seasoned" over sufficient time by finding those regions which are compressible, and which therefore presumably contain meaningful or interesting strings of symbols—that is, symbols with a linguistic or at least non-random basis.

Of course, that's just one way to process a universal library!

Another logical deduction is that half of the volumes or strings in this great maze will be mirror reversals of other strings. There may also be other, more subtle symmetries which our ambitious librarians can take advantage of to reduce their searches.

We could, for example, encode our library as a cubic construct of symbols, each coded in eight bits of data, arranged one hundred or one thousand to a side. There would then be rotational symmetries which could be anticipated and eliminated.

Randomness as a concept has became a foundation for many theories. Most famous perhaps is the notion that evolution can be accounted for only by accumulated point-by-point random mutations in a gene, through, say, heat, chemistry, or radiation. Such mutations are described in one textbook on evolution as "undirected."

John Maynard Keynes[2] refers to this sort of randomness as "objective chance," that is, random and unpredictable to any observer, real or imagined, anywhere, forever and ever. No observer will ever discover the "trick" or algorithm—or alternate theory—that could predict when, where, or how such a mutation will arise.

Subjective chance, on the other hand, supposes only that the relevant observers on the scene don't know of any trick to explain such events.

A pure version of variation in organisms caused solely by undirected mutation leads us to the inevitable conclusion that life itself constitutes a kind of processed and distilled "universal" library where meaningful strings are located through a winnowing process, that is, by survival within a selecting environment.[3]

A probabilistic examination of random mutations leads us down the same pathway of difficulty as any attempt to find meaningful strings within a universal library. How do undirected or random mutations in a string of, say, one hundred codons get "processed" to form a single competent gene? Between two and three hundred genes may be necessary for a minimal cellular organism. (This collection of genes will likely also rely on a pre-existing bubble or membrane of lipids—as well as the extensive and already-evolved amino acids, bases, and other molecules necessary for the function of any organism. But that's another discussion.)

The random recombinations of just one hundred codons would create sufficient numbers of molecules to fill a tremendous volume—terrifically larger in fact than our cosmos. And the time taken to generate them would likely outlast any number of universes. And yet many organisms utilize thousands upon thousands of genes of even greater length.

Evolution as a purely random process seems then to ride atop an immense Library of Babel, with one additional problem: our theoretical universal library is not allowed to repeat random changes in any string. Nature has no such rule. The time frame for the evolution of even the simplest and tiniest living organism, given these additional factors, becomes incalculably enormous. Ergo, random mutation is unlikely to be the unique force even in early evolution,

and less likely than *that* to be a dominant force *later* in evolution. (Though of course as a contributing factor—introducing stochastic "noise" into the system—it is quite significant.)

What can give supporters of randomness some comfort, however, is mutation in viruses. Where millions or billions of offspring are produced, and high rates of failure can be tolerated, random variation becomes a much more powerful force. Up to ninety-five percent of HIV viruses are defective, incapable of infecting a host cell, due to apparently random mutations. Such a high rate of mutation allows a scary venture into the unexpected creativity of randomness. Most viruses fail—but when they win, they can win big. But in larger, more complex organisms with far fewer reproductive opportunities, there are a great many corrective mechanisms that prevent unexpected novelty or error from going to term.

It's been estimated that one in three human pregnancies begin with twin embryos—but the mother's reproductive system in most cases selects out the fitter of the two embryos, and the other dies and is gradually resorbed into the wall of the womb. Such corrective procedures are common in nature, even after birth. Mutations that are not already fit, to one degree or another, are seldom allowed to re-enter the gene pool. These volumes are never returned to the shelves—so to speak.

Again, no corrections are allowed after the fact in a universal library. Genes have no such constraint. They are corrected and repaired on a regular basis, often turning defective or even so-called nonsense genes into competent players in the genome. As well, proteins assembled from errant genes are tested and if found wanting, disassembled into reusable amino acids. Deleterious mutations are thus usually weeded out before they ever have an effect on the organism. This is very like setting a great flock of editors before the printing presses in the universal library, where they discard nonsense volumes before they ever reach the shelves.

Random mutations are not always tried in the court of nature. That by itself calls into question the theory of undirected variation.

This purist vision has in recent decades had to beat a steady but stubborn retreat, not so much because of any mathematical objections as the discovery of alternate causes of variation—many of them far more likely to lead to increased fitness. The genome, it seems, contains its own highly sophisticated and cautious editing functions.

And those editors seem quite capable of responding to the biological "marketplace"—an ever-changing environment. Whether a tree blossoms at a certain time, or whether a bird changes its colors from white to dun, depends upon estimates of the "marketplace" at least as sophisticated—and with far more dire consequences—as those made by human bond or commodity traders. And similar natural forces may be at work in both instances.

Variation many times seems to pick from a pre-assembled tool-kit of both chemical and phylogenetic possibilities, many tried before and pre-approved, so to speak, by the genome's, or the species', rather astonishing internal editorial board.

As metaphor, the idea of the universal library could have contributed the perspective necessary to dispel an inadequate component behind a very compelling scientific theory. The idea of

the Library of Babel is certainly well known—but that seems far less important to the recent progress of biology than the extraordinary advances in biology in general, and in genetics in particular. And perhaps it should be so.

Still, the unique role of random, undirected mutation of genes—genes regarded as heritable traits—has for decades been considered so logically compelling, so irrefutable, that researchers who sought other explanations were severely criticized by their peers, their mentors, and by most senior scientists. Yet the idea behind such objectively random mutation is both philosophically and mathematically suspect.

Randomness as an etiological explanation has been a matter of faith in the sciences for almost a hundred years. With regard to the neo-Darwinian synthesis, a great many senior scientists still firmly believe that only random mutation will ever explain variation. They cling to this in the face of a great many counter examples—not all of them recent.

Meanwhile, the expanding study of epigenetic variation—which describes how groups of genes are turned on and off during development—adds to the amazing process of how living organisms reproductively adapt to their changing environments. If we regard the genome as a piano keyboard, epigenetic variation is like a composer deciding which keys will be played—and what melodies will result.

It's becoming more and more clear that genuinely random mutation in genes contributes a much smaller percentage of observable variations in wild-type populations of complex organisms. Nevertheless, it is still the academically dominant metaphor—the paradigm. And that paradigm, while conferring some discipline to a younger science, has too often held back discovery of the actual mechanisms by which organisms evolve.

Once remarkably fit and vital, random variation now needs to be edited, judged, and re-shelved in the library of observed reality.

Endnotes

1 Gregory J. Chaitin, "Randomness and Mathematical Proof," Scientific American 232, No. 5 (May 1975), pp. 47-52)

2 A Treatise on Probability, Macmillan and Co., London, 1921; page 282. Keynes traces the notion at least back to David Hume.

3 An interesting comparison between these two problems was made by Daniel Dennett in 1995 in his book *Darwin's Dangerous Idea.*

Seven Habits of the Highly Effective Rick Rashid

Terri Rashid

I have known Rick Rashid for over two decades, and we've been married for fourteen years now. During that time I have had the chance to see how he handles a wide variety of situations. I have always been impressed with his skills as both a scientist and a leader, and while I am obviously not without bias, I believe he is one of the best managers I have ever met. His success in building Microsoft Research from scratch to the world-leading computer science research lab that it is today is an objective testament to his outstanding technical and management skills.

In this paper I share some of the insights that I've gained from watching Rick manage both people and situations. His personal style of being positive and optimistic, mentoring the people around him and recognizing their accomplishments, and his excellent self-control, discipline, judgment, and confidence continually inspire me to try to strengthen my own interpersonal skills and strength of character. It is difficult to convey these impressions in a few pages of writing, but given the impact they have had on my life I believe it is worthwhile to try.

As I began to write this paper, I was struck by how many of these ideas seem fairly obvious in hindsight, or have been written about in some form or another. While these realizations have definitely improved my own leadership skills, I was uncertain how a paper by someone who is not an expert in this area could be really effective. And yet, as I thought about how many times I have needed to remind myself of some of these strategies when I am composing email, or managing a difficult conversation between people with opposing views, it occurred to me that important lessons bear repeating. Again. And again. And again.

Rick and I are currently learning to fly small airplanes. There is a particular lesson that is actually taught, which sounds incredibly obvious if you write it down. The first rule of aviation is "Fly the plane." No, seriously, they *teach* that. Sure, it seems obvious, but you'd be surprised how many accidents occur because a pilot became distracted and forgot that

most obvious point. So it's repeated by our instructors as we progress, and especially as they are teaching us to handle cockpit distractions. And that repetition works.

I was flying right-seat in a slightly larger plane with an experienced pilot when we were hit by a downdraft that was hard enough to send Rick and the boys up hard against their seatbelts and a non-essential (passenger) radio panel to be yanked out of the cabin wall. I looked over my shoulder to check on Rick and the boys, and when the pilot looked back as well to see what had come loose the first words out of my mouth were, "It's OK, fly the plane." In hindsight I was a bit embarrassed to have said that aloud, but I was concerned that we might be hit by another hard gust and I didn't want him distracted by what was going on in the back of the plane. Still, it shows that through repetition, the first thing that I thought of was "Fly the plane."

Amusingly, in parachuting, there's a similar essential lesson: "Pull." Meaning, no matter what else happens, you need to remember to pull the ripcord of your parachute. Seems obvious, but it's still taught. The longer form is "Pull. Pull at a safe altitude. Pull at a safe altitude while stable." But the overriding priority is to PULL your ripcord, and they repeat that throughout training.

So even if some of the ideas presented here seem a bit obvious, I think they are worth repeating. While the act of managing in our field, unlike aviation decisions, is generally not a life-or-death situation, the actions that managers and co-workers take can have a huge impact on the lives of those around them.

I want to include an important disclaimer here for clarification. Many of these strategies are not things that Rick has taught to me, or told me, they are merely my impressions from observing and interacting with him. He might not even agree with some of the points I present here. As most teachers know, what someone intends to teach is often not the lesson that the student learns. Rick's interpersonal skills are something that he does naturally; with a few exceptions they are not really the subject of conversation between us. So with the understanding that these are my interpretations and not Rick's, I hope that you find some of these useful in your own life.

Give Credit, Don't Take Credit

I am starting with this idea because it is the one that I have personally seen make such a huge difference in group interactions and I don't think it is as well-understood as some of the others.

It is natural for people to want to be recognized for the work that they do. Numerous articles have been written on the factors for personal and employee satisfaction and recognition and respect are often listed as top contributors. So it's not surprising that, especially in a competitive situation, people may feel the need to make sure that they get credit for their ideas and work. Unfortunately this is often done by trying to *take* credit for something, which can cause hard feelings if someone else feels that credit is being "*taken*" from them.

From what I have seen at Microsoft and other organizations that I have been involved with,

it is much more effective for all concerned to focus on a pattern of giving credit. When your co-workers do something interesting or develop a good idea, recognize them for that and share it with others. When people are in an atmosphere where credit is given, they stop worrying about the need to take credit because no one is going to give it. As their confidence that they will be recognized for their contributions grows, they also start to give credit to their co-workers. In this situation people share more freely and are less defensive. Since giving credit typically generates good will, people feel good about the credit they have given, as well as the recognition they have received for their own work. This doesn't mean you need to give credit for nothing; that would simply de-value the recognition.

The tricky thing about this lesson is that it requires confidence and security for someone to initiate. Rick is one of the most confident people I know, so this seems to come easily for him. If someone is worried that they are not going to be recognized for their work, they typically find it more difficult to recognize others without trying to take full or partial credit for themselves.

Another important aspect of this as a manager is to be aware of not just the general work that someone is doing, but know enough about the specifics to be able to really understand the challenges that were overcome to achieve the results, both to be able to congratulate the person doing it, and convey their accomplishments to others. I have been in numerous meetings where Rick not only gives credit to groups in Microsoft Research, but also makes a point of highlighting work done in groups outside of MSR that contributed to or facilitated those projects.

Be an Optimist

In our family we often joke about Rick being an incredible optimist. This shows up in amusing traits, such as what we refer to as the "Rashid Parking Karma." Rick will end up with front-row parking, even at a crowded mall or theater, way more often than seems believable. Part of that, though, is because he is an optimist. When I see that a parking lot is crowded, I will typically settle on the first halfway-close spot that is open. Rick, however, will continue up to the very front, because he optimistically believes there might be a spot open there. And, not surprisingly, this means that he will get a front-row spot many times when I would not.

From a management standpoint I've seen this trait realized in a couple of ways. First, Rick is always aware of, and looking for, opportunities. He is willing to consider things that other people might think are too hard, because he is an optimist. That doesn't mean that he isn't realistic in his evaluation of various problems, just that he is open to a wider range of initial possibilities for consideration.

The second aspect of his optimism is how he reacts to adversity. He truly believes that when a particular opportunity does not work out, there will be something even better to come along instead. This is his fundamental optimism at work. It has taken me years to really internalize this, but he's been right so many times in the past that now I believe it almost as much as he

does. Inherent in this attitude is the choice to see the new opportunities surrounding you, even if they were not the ones you originally pursued.

This past year two of my horses become injured shortly after we moved to Arizona. One of them was my elite competition mare. This meant missing a major national competition last year. I was disappointed, but used Rick's philosophy to look for other opportunities. I decided to focus on my running again and last fall managed to run my fastest marathon in 12 years. And even more importantly Rick and I were able to spend a large amount of time together in August learning to fly. We both enjoyed the chance to learn something completely new, and do so together.

Focus on Positives

This idea has some similarities to the optimistic attitude described above, but applied in a slightly different context. The strategy here is to focus on positive points and descriptions when trying to argue a point or convince someone to your position. This becomes even more important if the issue is emotionally charged and the discussion is in a group setting, either email or face-to-face.

As academics we are taught to be critical thinkers and readers. When reviewing papers, one of the important tasks is to be alert for flaws in reasoning or methodology that might invalidate the authors' conclusions. So it is not surprising that many times our first approach in conversation is to point out how an idea we disagree with is wrong. Unfortunately, when this technique is used in a discussion of a sensitive or polarized topic, the person whose viewpoint is being criticized often feels personally attacked. This is particularly true in email where the defensive reader infers a tone of voice for the writer that may not actually be intended. I've seen too many examples of how this can quickly devolve into a chain of messages that lash out with personal insults that are not even remotely related to the topic of discussion, but simply occur because of defensive emotional responses.

I think Rick is particularly skilled at diffusing such situations through calm, reasonable messages that focus on the positive objectives and/or highlight a high-level common goal. He is very adept at recognizing and avoiding comments that are likely to be interpreted as antagonistic and his self-confidence and control mean that he does not let intentional baiting affect him. (This is not to say he never displays irritation in email, but in general I believe he does so judiciously and when he intends for it to have a desired effect.)

Through his example, and also his helpful pre-review of many delicate email messages over the years, I've developed a set of strategies to help make my email (and conversations) less adversarial:

Eliminate words that are inflammatory or derogatory, as well as sarcastic remarks. In my experience this step is the easiest and accomplishes the most benefit for the least work. Saying that an idea is idiotic or stupid may make you feel better (I often say this to Rick in private for the therapeutic effect), but it does very little to win other people to your point and it will almost certainly alienate the original author. Pointing out their previous failures is similarly

unproductive. Depending on the situation I even try to avoid the use of the word "wrong". If needed for clarity I might say that I disagree with something, but often that is apparent from the discussion. If the topic is really important, and you are trying to build consensus, consider having a non-involved friend/co-worker read through the email before you send it. They may spot things that you didn't really think were abrasive since they are able to read with a more objective view.

*If possible, instead of arguing **against** another person's idea, or pointing out the flaws in it, make your case by arguing **for** an alternative idea.* State the positive attributes of the approach you advocate. There is a tendency to compare and contrast ideas, but in a sensitive discussion you may be better off only stating the advantages of your idea and leave out the corresponding flaws in the alternative. Your audience will probably be able to infer those for themselves. If the drawbacks are not obvious and you feel it is important to ensure that people understand them then try to voice them in a non-confrontational manner.

This area is one where I have to be particularly vigilant. I have a tendency to use words like "obvious" or to state opinions as facts. Rick has pointed out how these kinds of statements can be antagonistic even when that is not the intent. Expressing something as a personal opinion, instead of trying to assert it as a fact that should be apparent to all, will get the point out to your readers without implying that anyone's intelligence is in question. E.g., "I am concerned that if we do this [some non-optimal thing] might happen," instead of, "If you do this [some dread world-ending thing] will happen." The former statement expresses an opinion, is inclusive by saying "we"—fostering a team attitude, and does not over-exaggerate the consequences or assert them as a foregone conclusion. But it still highlights the potential downfalls to people who are actively trying to evaluate the situation.

Finally, if there are parts of the other person's idea that you think are useful and worthwhile then say so. If you can bring the focus to things that you agree on and have some positive interactions (e.g., *Give Credit*), then you stand a better chance of persuading them on some of your other points.

Just as with giving credit, the practice of focusing on positive dialogue has its own set of difficulties for implementation. It can be extremely challenging to forgo insults and sarcastic comments, especially if the previous author directed such comments at you. The entertainment industry uses cutting insults and witty sarcasm to great effect in books and movie dialogue. They are often considered a mark of quick verbal intelligence. I personally find some of those exchanges very entertaining and have to work hard to exclude clever sarcastic comments that highlight amusing flaws in an opposing author's viewpoint.

But it is important to consider the goal of the discussion. Generally, if I am bothering to participate in such an email debate, it is because I care about the outcome. While I might entertain someone with such remarks, it is probably only going to be the people who already agree with my point. I have found, time and time again, that by ignoring the insults and jabs in an email discussion, and simply replying to the actual content, more people have been persuaded to my view. Sometimes people who didn't even have an opinion on the topic

itself will decide that I am probably right because I am clearly approaching the discussion in a more rational manner.

This is not to say that you can't be passionate about the point you are making in a debate. On the contrary, that is often highly desirable. If that enthusiasm is used to generate excitement about the advantages of your approach then others will feel more positive about joining your view.

Mentor & Delegate

Rick has often said as he built Microsoft Research that one of his key philosophies was to hire great people and then enable them to do their research. This philosophy has worked exceptionally well at Microsoft Research. By hiring great people, not only do those people do great research, but they can also grow into great leaders for their own groups, increasing the effectiveness of the entire organization.

But Rick doesn't just hire great people, he also takes the time to mentor them and encourage them to mentor others. In addition to traditional one-on-one meetings, he will frequently drop by someone's office for a quick chat. His manner when he's on a hallway stroll is more relaxed and people who know him recognize that and use the opportunity to ask him for a minute or two to discuss something. At lunch he likes to sit with different groups to catch up on what is happening and tell stories in a relaxed social setting. His stories, drawn from both his time at Microsoft as well as experiences as a professor and back to his college years, often illustrate some particular point with a humorous and memorable twist.

Even people that naturally have good leadership or interpersonal skills benefit from being mentored. Sometimes the mentoring might help them have a better understanding of how different groups across the company interact, or how the business priorities for a particular group might affect their ability or willingness to integrate new research technologies. And people who are starting to move from the role of independent researcher to group-leader make that transition more smoothly with a bit of guidance.

This combination of hiring great people and mentoring them, results in people that manage their areas very well, and can be trusted to handle delegated tasks in an intelligent and independent manner. Rick is very good at encouraging people to make their own decisions and stand by them. In the spirit of mentoring someone he will often give examples of how he thinks about certain issues, or what kind of approach he might use to solve a particular complex problem, but then he will typically encourage the person to apply such techniques themselves and make their own decision.

Reasonable Person Principle

Carnegie Mellon University, where Rick was a professor prior to starting Microsoft Research, encourages the "Reasonable Person Principle" among faculty and students. The general idea is that if each person behaves in a reasonable fashion and considers the needs and goals of others and the school when making decisions, then there is a greatly reduced need for arbitrary

and complicated rules to enforce specific behavior. Rick brought that philosophy to Microsoft Research and encourages people to use it in their decision making.

It is ironic that if there is a specific rule to try to constrain behavior, many smart people will reflexively look for the loophole that allows them to get around the rule. The very presence of a constraint seems to provoke a desire to be free of it. Unfortunately, the result of people finding a way around these rules is that the rule makers are then prompted to introduce more constraints in order to prevent those behaviors. At some point, there are so many constraints that sometimes people are prevented from doing perfectly **reasonable** actions that were not foreseen when developing the rules.

Budget, headcount, and expenses are areas where this tension is particularly evident. Faced with constraints (limited resources) the focus sometimes gets shifted from, "What do I reasonably need in order to accomplish this task / support my group?" to "Given this set of rules, how much can I get?" The latter is not reasonable and does not really take into account others' needs and the needs of the company. In general Rick asks people to be reasonable about their headcount and expenses. This allows for the most benefit across the organization. Certainly tradeoffs still have to happen at each management level, but options are not artificially cut-off because of resources being held where they are not currently needed. The more effectively this culture is promoted (both through example and mentoring), the fewer explicit rules are needed, preserving greater flexibility for everyone.

The greatest benefit to this approach, even beyond the organizational advantages, is that it creates an environment where people feel trusted and respected and which fosters their active consideration for the people around them, resulting in more positive interactions. In my experience as a student at CMU I know that I greatly appreciated the latitude that was given to us. We were told that the administration trusted us to be reasonable in our actions, and that prompted us to demonstrate that their trust was not misplaced. During my time there I felt the system worked remarkably well.

I realize there are practical limits to the application of this in a large enterprise, but the philosophy is sound and I think it should be applied as widely as possible.

Discipline, Decisions, and Time Management

I must admit I truly envy Rick's ability to simply decide that he's going to do something at a particular time, even if it is something he doesn't enjoy, and then sit down and do it. There are lots of tasks I dislike (anything tax-related springs to mind) and I will often procrastinate and leave something hanging over my head for days or weeks before finally getting it done. Sometimes I will even decide that I am not going to do something at all, after procrastinating sufficiently long that it's either no longer feasible, or just no longer important. Or I will do a rushed job because I've left it until the last minute. While it might seem that there is some benefit, then, in waiting to do these things, since in both of the latter cases the resulting inaction took less of my time, my experience tells me otherwise. The stress of having something hanging over your head is not pleasant or particularly healthy.

This does not mean that you have to do every pending item immediately before you can do anything new or take some time for yourself. I've seen how Rick typically handles unscheduled tasks that require some focused time. Examples might include preparing a talk, reviewing the budget, or preparing for employee reviews. He knows when these things need to be completed by and he will have a mental idea of when during the week he's planning to work on them. He doesn't schedule specific times for things like that as a rule, but he keeps an eye on his schedule and knows when he plans to deal with them. If he has some unexpected free time on his schedule, for example because of a cancelled meeting, then he may use that opportunity to accomplish one of these tasks, or he might choose to have an impromptu hallway conversation with another researcher. He has an excellent knack for balancing both structured and unstructured time in productive ways.

A key component to this is that Rick has a very good grasp of how long he will need to accomplish various tasks. Some of that is through experience, but some of it is also by decision. I have seen people who take important tasks and allow them to consume every non-scheduled minute of each day until the task is due. For most tasks this is not effective. Whether writing a paper or preparing a presentation, you should have a reasonable expectation of how long it will take. By planning to spend roughly that much time working on it, you will focus more on the important parts of the task and be less likely to waste time simply because you have it. Additionally, you will have enough time to do a good job. Plan to have the task complete before it is due. Then when an unexpected situation arises that requires immediate attention you have the flexibility to deal with that without the stress of missing a deadline or having to work until all hours of the night.

Rick is also quite adept at deciding which items require his attention and which do not. But from what I can see he makes that decision up front. He doesn't leave something hanging over his head until there's no time left to do it. He decides if it needs to be done and if so, allows an appropriate amount of time for it, and if not doesn't worry about it.

The combination of these traits—the experience to know how long something should take, the ability to decide what is important and what is not, and the discipline to deal with items at specific times I believe is part of what allows Rick to manage a large amount of complexity without a high level of stress. As someone who greatly enjoys our free time together I can say that it also means that when he's spending time with me and/or the boys he is able to be joking and engaged with us, even when there are difficult decisions pending at work.

Don't Worry…

As implied in the previous paragraph, Rick is not a worrier. I can't count the number of times that I have heard him say, "I'm not worried." Sometimes this is not a welcome phrase when it is said in response to me telling him about something that I *am* worried about. However, as usual, his approach is very effective. He doesn't worry, but he will make a plan to address an issue if that is needed, or possible. Once he has done what he can he will move on to something else. If the situation is something he can't affect or control then in general he really doesn't worry about it and will again move on to other things.

This technique ties back into a number of the other points that I have mentioned. When Rick is making a decision about something, he considers the relevant information, makes the decision and moves on. If he has decided to delegate something to someone, he does so, and moves on. He may also mentally make a plan for when to check back in with them, but he won't continually worry about what they are doing.

To me there is an important distinction between worrying about something and considering it. Worry is an emotional process where I continue to replay the same concerns in my mind, and possibly weigh some alternative actions, but I also spend a lot of non-productive time mentally reviewing the same things repeatedly without taking action. Or having taken action, waste time second-guessing myself after the fact. In contrast, when I *consider* something, I gather whatever information I need, or at least what is available in a reasonable amount of time, consider it and then make a decision.

The caveat, "in a reasonable amount of time," must be interpreted based on the importance of the decision and the frequency with which such decisions arise. If you need to handle a dozen such items per day then you certainly can't spend several days on each one. Make a decision and move on. If the decision affects an entire group's future then it is worth more time to collect relevant information if possible. But even in that case it is not useful to "worry." (It may be unavoidable for someone like me, but it's not useful.)

…Be Happy (Conclusion…)

OK—this is really the important point. Most of us would like happiness for ourselves, our families and our friends. While I don't think there is a simple solution for finding happiness, I do believe that positive interactions facilitate happiness and satisfaction.

Giving credit to someone is a positive experience. I feel good when I can recognize someone else and make them happy. Trying to think of positive points in a discussion or ways to get people enthusiastic about an idea first requires that ***I*** think about those positive things myself. Mentoring people is very rewarding, both in terms of the personal relationships developed and the practical aspect of having people you can rely on. The Reasonable Person Principle encourages an atmosphere of mutual trust and respect, an environment that is inherently rewarding. Practicing self-discipline, especially as it relates to planning decisions and time management reduces the stress of managing complex or large projects. Focusing on actionable concerns and not using time in emotional, non-productive worry again reduces stress and increases effectiveness.

So all of these strategies, in addition to the specific benefits they confer, also generally contribute towards positive interactions and situations, and I think, happiness and personal well-being.

As I said at the start of this paper, many simple lessons are conveyed best through repetition. But unlike my simply re-stating and repeating them here, I have had the benefit of seeing the effectiveness of these skills and strategies in my observation of Rick in dozens (hundreds?) of different scenarios and situations, and through his guidance when I ask for input. I am fortunate to have as my best friend and life companion someone who not only inspires me

to try to become a better person but facilitates that process by being an excellent mentor and role model.

As those who know me well realize, there is plenty of room for improvement and I definitely still have a lot of work to do in many of these areas. But I have personally used each of these techniques in various situations and have seen and felt the positive results. Rick continues to help me both directly and by example, and while these strategies may not be exactly habits for me ***yet***, life is a journey and I am doing my best to direct it along the path I intend.

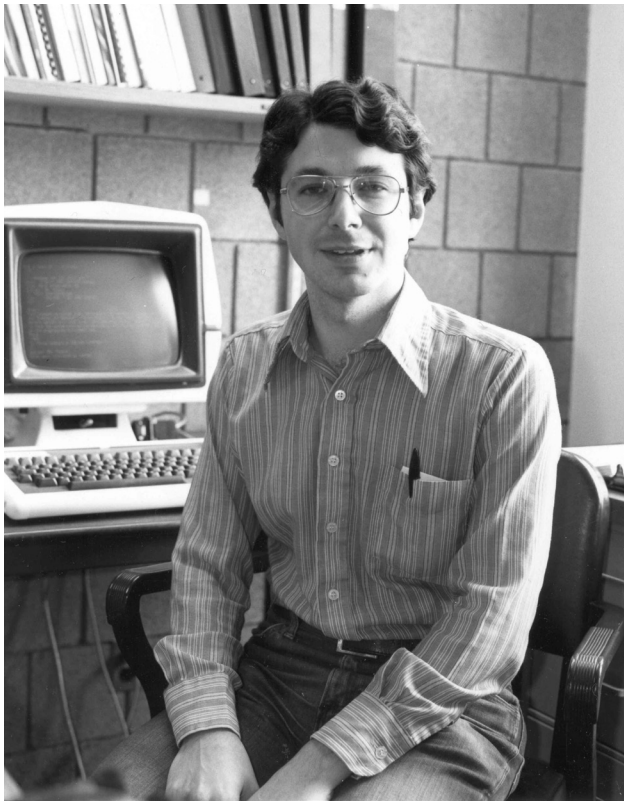
I wish you all well on your journeys and hope you find happiness and satisfaction.

Terri Rashid

I would like to thank Astrid Bear for providing wonderful, constructive, and often times humorous, commentary on the draft versions of this paper; it was definitely improved through her feedback. Remaining weaknesses are strictly the fault of the author.


hotographs

Early Years



Opposite page: Young Rick

Top Left: Rick in a Hawaiian Shirt some time before grad school.

Top Right: Rick pounding out code.

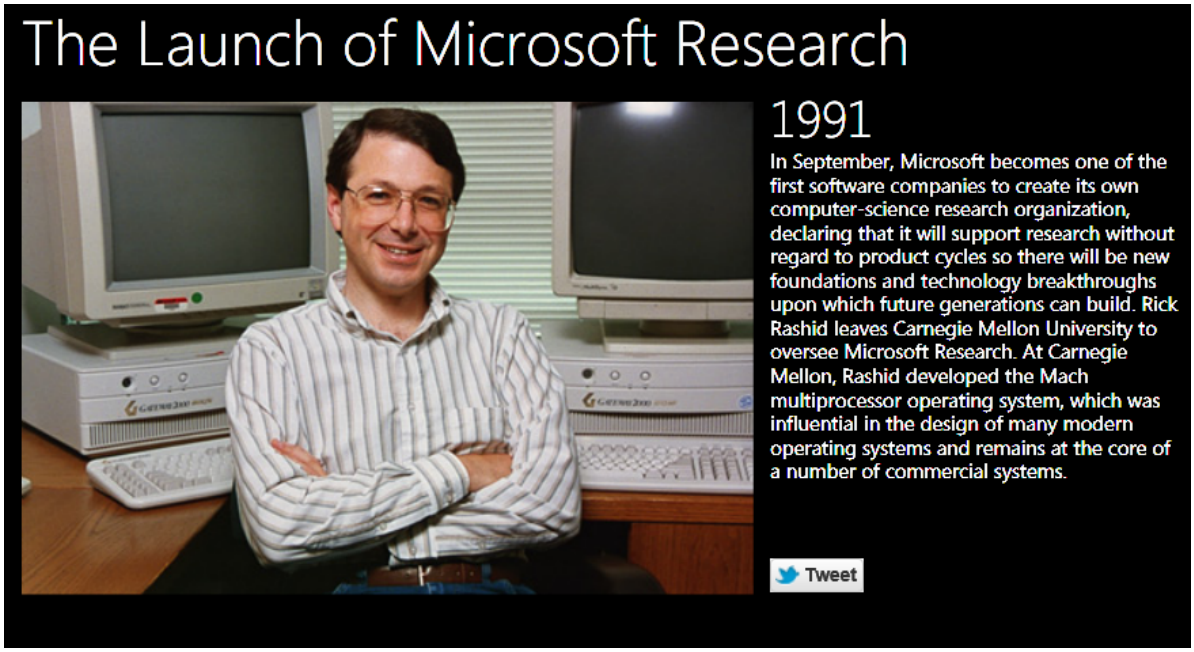
Bottom Left: Official photo for CMU CS Department.

Bottom Right: Rick's office in Wean Hall at CMU.

This page: Professorship and Beyond

Right: Rick at CMU before 1991.

Below: Timeline from Microsoft Research web site 2012.



Bill Bolosky, at left, running into Rick at SeaTac airport in 1991.

Staking a claim to a stretch of digital asphalt

BETTY UDESEN / SEATTLE TIMES

Rick Rashid, *director of research for Microsoft, explains "Tiger" to press and industry analysts.*

Microsoft showing its stripes with introduction of 'Tiger'

BY O. CASEY CORR
Seattle Times business reporter

It looked like an ordinary TV, but its development cost more than $200 million.

A man pointed his remote control and clicked.

"Movies," read the screen. Click.

"SciFi." Click. Then a "Star Trek" movie began playing, as Microsoft demonstrated its claim on a future marketplace called the information superhighway.

Where there are computers and money to be made, Microsoft is sure to go.

The bullyhooed information superhighway is expected to be a huge market. How big is anybody's guess, but the Redmond software giant plans to own the digital asphalt.

Microsoft yesterday unveiled its "Tiger" technology for controlling devices that store and send video images, text and sound to TVs or personal computers. Within two or three years, cable TV systems in Seattle and elsewhere will be able to provide hundreds of channels that allow users to "talk back" to the cable company or to other people.

Using Tiger, a viewer can select a movie, shop or play games with other people. Microsoft also thinks the technology will open up a whole new field of interactive entertainment ("Vote yes if you want Roseanne to reunite tonight with Tom Arnold").

Microsoft invited the press and industry analysts to see what has been costing the company $100 million a year to develop. To symbolize ongoing construction of the information superhighway, the room was decorated with scaffolding, yellow tape and people in hard hats. Sixteen

PLEASE SEE *Tiger* ON D 2

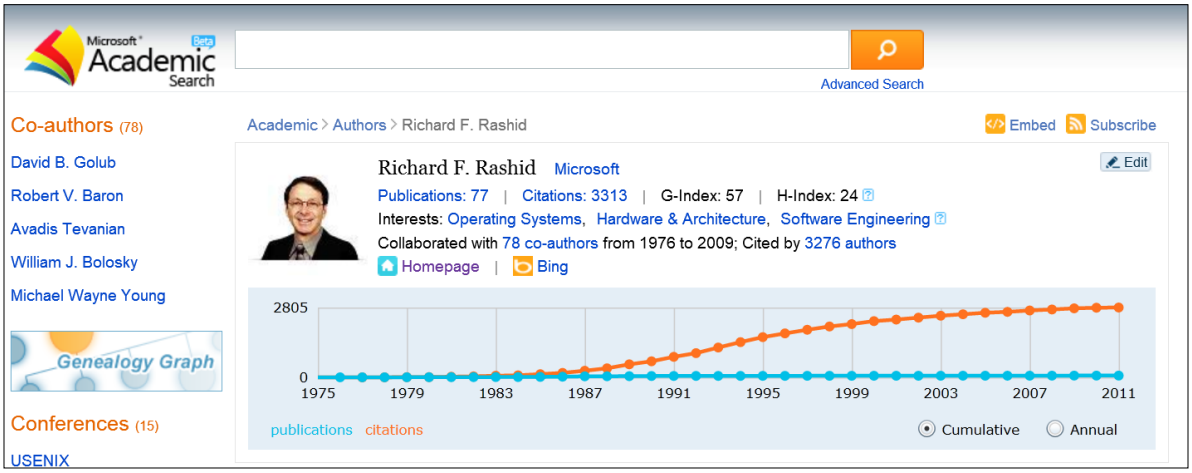
SEATTLE TIMES 5/18/94 PAGE D1

May 17, 1994: Rick introduces the architecture for Microsoft's new software solution, code-named "Tiger," for delivering continuous media.

Rick sticking his head through the Great Wall of China, October, 2006.

Rick addresses the crowd at TechFest 2012, where his "Star Trek" inspired language translator is announced.

Rick received the Technical Recognition Award for Career Achievement in 2009.



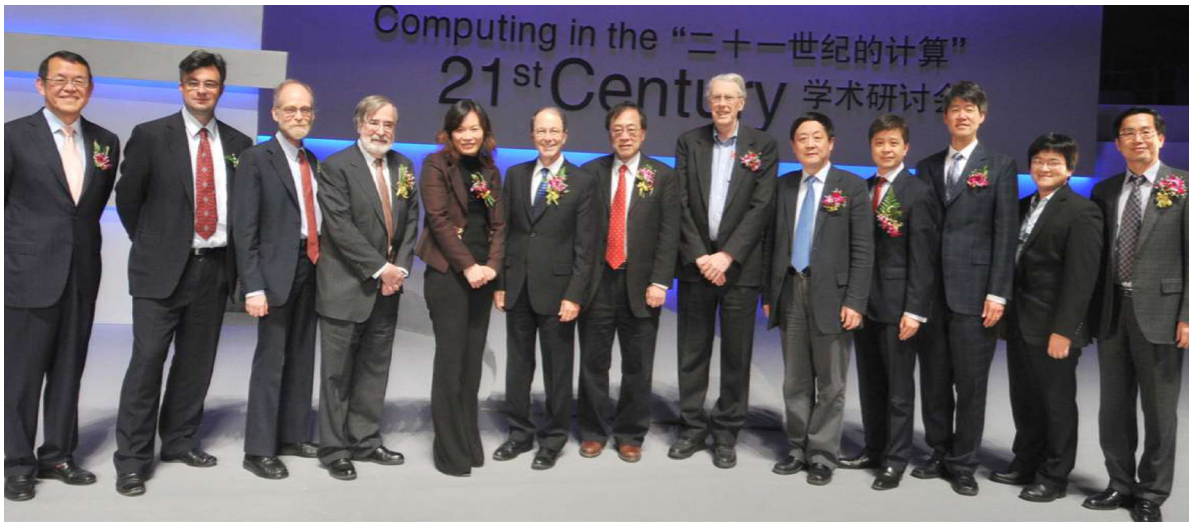
Rick Rashid's profile on Microsoft Academic Search.

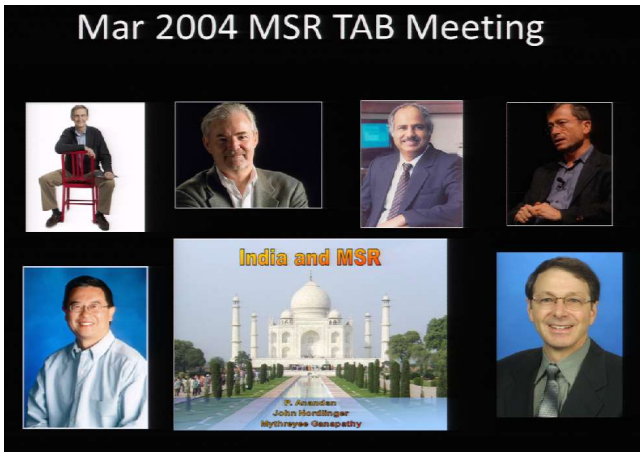
Top Left: Rick in front of the Taj Mahal during the startup phase of MSR India in Banglaore

Top Right: Tsinghua University, Beijing, 2006. From left: Rick, Eric Grimson of MIT, Ed Lazowska of the University of Washington, Pat Hanrahan of Stanford University.

Center: Christian Borgs, Jennifer Chayes and Rick at MSR New England in Cambridge, Mass.

Bottom: Computing in the 21st Century, Beijing, 2011. From left: Victor Zue from MIT, Emmanuel Candes from Stanford, Joseph Halpern from Cornell, Edmund M. Clarke from CMU, Qian Zhang from HKUST, Rick, Andrew Chi-Chih Yao from Tsinghua university, John Hopcroft from Cornell, Jianping Wu from Tsinghua university, Hsiao-Wuen Hon, Peter Lee, Xi Chen from Columbia, Wei-Ying Ma from MSRA.





From an interview with Bobbie Johnson, The Guardian, UK. February 2009

Above: The 2004 Technology Advisory Board Meeting in Bangalore, India. From bottom left clockwise: Dan Ling, Ed Lazowska, Richard Newton, Raj Reddy, Andy Van Dam, and Rick Rashid.

Right: Rick and Terri at the National Academy of Engineering Awards, October 2003.

Below: The Rashid Family: From left: Tieran, Kira, Kylan, Terri, Rick, Danny, Farris.

From all who have known, befriended, and learned from you, for all you have inspired.

May you continue, Rick, to boldly go where no one has gone before!