

A MAGAZINE FOR AIRLINE EXECUTIVES

2010 Issue No. 2

ascend

Taking your airline to new heights



A TOP CONTENDER

A Conversation With ...
Enrique Cueto, Chief
Executive Officer, LAN
Page 12.

18 Cambodia has a new, proud national flag carrier

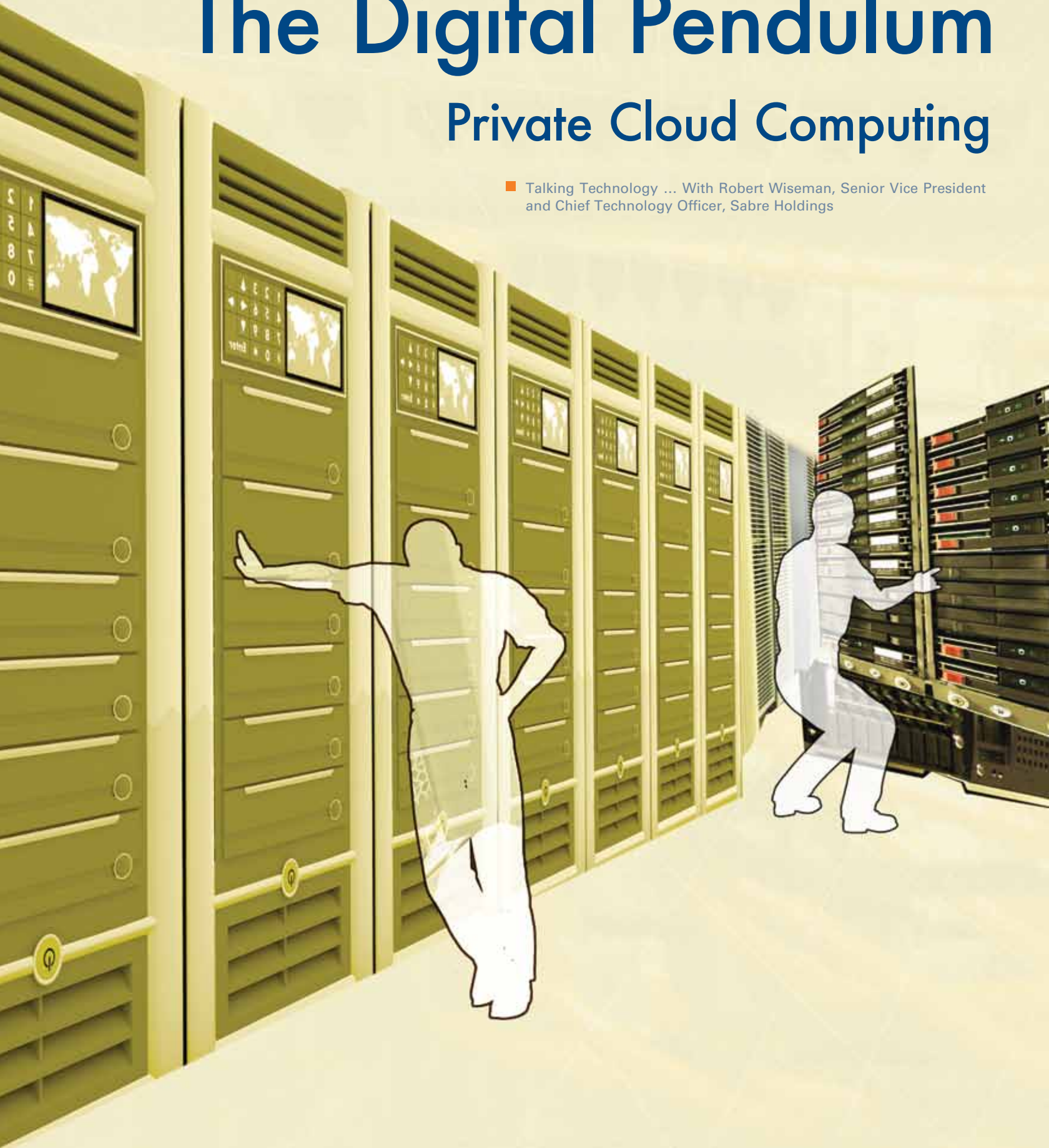
41 A new era in airline technology is upon us

76 single, robust platform

The Digital Pendulum

Private Cloud Computing

■ Talking Technology ... With Robert Wiseman, Senior Vice President and Chief Technology Officer, Sabre Holdings



I understand that computers existed before I first worked with them, but in the scale of time, certainly in the scale and pace of computer time, it doesn't seem as though it was that long before.

In 1978, I was still in my native England, working at a drill company in the northern city of Sheffield (where the movie "The Full Monty" was set), at the time, the steel capital of the world.

My home was a farming town 15 miles away. This was a great distance by English standards, at least by 1970s English standards, which amazed many of my city-dwelling workmates who would often ask me how I could stand to travel so far each day.

In truth, it wasn't an easy journey: two buses and a mile walk to get there in time for my morning shift as a computer operator. The shift started at 7 a.m., so I had to leave my house by 5 a.m. to get there on time. But waiting for me at the end of each epic journey was one of those wonderfully mysterious "computer things." A refrigerator-sized, black, brand new IBM 370-135 — and that always kept me going.

It didn't have as many flashing lights as I'd first imagined it might, but this was more than made up for by the fact that it had the first CRT (cathode ray tube) I'd ever seen plus several mechanical cabinets that housed those whirring, menacing tapes that never seemed to be able to decide which way to spin.

My first role as a computer operator consisted of a number of standard manual tasks — mostly feeding the giant and voracious chain-link printer that seemed to eat its way through endless boxes of green-striped, three-ply carbon copy paper, which frequently jammed as it sped through the printer's violent, clanging machinery.

Once the people there decided I wasn't a total country bumpkin, I was allowed to touch "the keyboard" to actually control what the computer did — more or less. To be honest, back then at least, operators' skills relied mostly on their ability to follow a simple script that was printed out each morning with instructions such as: when you get this question, always type this; when you get this question, always type this; when you get this question ... you get the idea.

About seven months into my job, I was made shift leader (it was a very small company) and immediately looked at ways to automate the process so the scripting tool (job control language) would answer its own rhetorical questions. This resulted in very smooth and, frankly, rather more boring shifts. Three months later, either

impressed by my initiative or annoyed that I'd created a surplus of operators (possibly a little of both I suspect), they moved me out of operations and into the programming group where I was taught to program reentrant Assembler (a type of Assembler that allows its code to be executed simultaneously by multiple processes, meaning that any switch/flag changes need to be made outside of the program boundaries). Maybe it was a punishment after all.

In the 1980s, I moved to America, still working on IBM mainframes, still coding Assembler, trying to learn as many details and nuances as I could about the mature and seldom-changing environment.

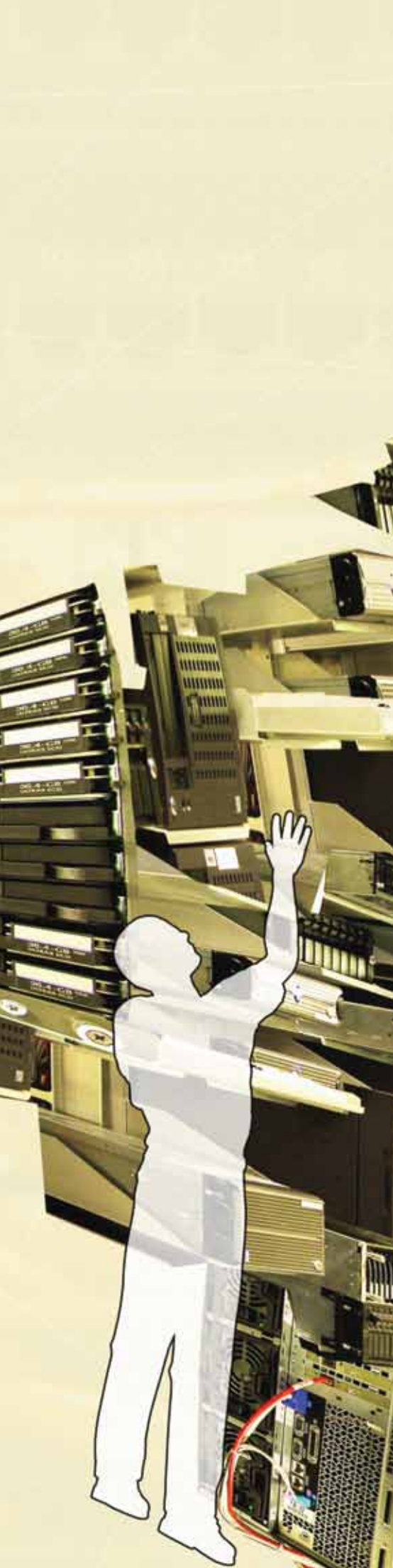
Then in the late '80s and early '90s, a big change, called client server, hit the computing industry. Almost at once, every new product had to have its own server hardware, database, fat-client application and dedicated team of specialists to maintain it. It was around this time that I first began to move out of the mainframe world and into this "new" and confusing world where, it seemed, everyone had a different way of doing what we'd been doing all along on "big iron." Processing data!

I was definitely taken by the cool presentation capabilities of the newer technologies that were vastly superior to their mainframe/green-screen counterparts. Even today, though, mainframe interfaces are still surprisingly popular because of their simplicity — this is even true in places where customers have options between text and GUI — despite being a tad clunky and severely disadvantaged in what they can display.

The "dumb terminals," as they were aptly named, had neither sufficient processing power nor "imagination" to participate in anything other than an exchange of **RUDIMENTARY DIALOGUE**, which is how it appeared "back in the day."

The other thing I really liked about the newer technologies was their low start-up and scaling costs. Now, for the first time, companies could get off the ground with just a few thousand dollars of hardware — and when the Internet revolution arrived, and brought with it another wave of technology that would challenge the burgeoning client-server movement, that's exactly what happened.

The biggest concern I had with how client server was evolving, at least in the company where I worked, was the unchecked sprawl. So many vendors, chip technologies, hardware models, operating systems, databases, languages, test tools, etc. The pendulum, it seemed, had taken a long and arching swing away from monostandard to hetero or totally non-standard and, although far less expensive to acquire,



much of the compute power (very much if you include the “client” hardware as well), was grossly underutilized, resulting in wasted capacity, power, space, cooling and capital. Although probably a slight exaggeration, it now appeared as though no one was sharing — initially because of a lack of planning and controls and then eventually because respective system incompatibilities and lack of established standards prevented them from doing so.

When the Internet revolution came, it brought with it many valuable attributes — and in the context of this discussion, two really stand out:

1. The Internet browser. Initially, most people saw this as just an external, consumer-facing tool and kept marching right along with their client-server plans. But early on, a smart colleague of mine asked, “Why wouldn’t we develop browser solutions for professional agents as well as for consumers?” Others, too, began to question: had the pendulum swing from “dumb terminal” to “master terminal” gone too far? Soon, the time and cost of maintaining and deploying thick-client apps — almost exclusively via floppy disks or CDs back then — forced a swing correction to the mix of presentation solutions we have today: full-client-based applications (MSWord); rich-client applications (Google Earth and *MySabre*[™] agent booking portal) and an entire range of browser applications with a variety of thicknesses measured by the number and size of downloads (Flash) needed to execute them.

2. Horizontal scalability. New-entrant companies, that had missed out on the “American Idol” of computer technologies in the early ‘90s, were now able to select from a narrowed field of fast-maturing final contestants. This largely meant that they settled on a single, low-cost “pizza-box” solution, with the hopes of scaling it out for low-cost processing, targeting stability through redundancy and trying to get as much reuse as possible. This was the right plan, of course, but scaling horizontally is much more difficult than “people” — that is those odd groups of us who tend to think about these kinds of things — tend to think they will be. It takes a lot of planning and coordination — and it requires telling developers how they need to develop their code. And as anyone knows, developers don’t like being told how to develop their code. I was one once — and I know.

Good design practices, such as service-oriented architectures (SOAs), go a long way in terms of enabling the ability to scale across thousands of servers, but we need much more than just “good



During his first four years as chief technology officer for Sabre Holdings, Robert Wiseman moved the company to its first enterprise-wide set of infrastructure standards, which are managed by his organization. He has 32 years of experience in information technology; 24 of which have been in the travel industry.



practices.” At the types of volumes we run at *Sabre Holdings*[®] — 32,000 transactions a second, operating globally, 7x24x365, with agents and airlines of all sizes, dependent upon our systems for their livelihood — our solutions have, to say the least, high demands on them.

We strictly enforce well-proven architectural tenets in all of our systems — tenets that improve the resiliency of our software. We validate the implementation of those tenets every day in our high-performance labs. We ensure that they will scale and can withstand volumes at far higher rates and for far longer periods than they will ever experience in production. We also do this with third-party products. If they can’t survive our scalability “boot camps” — and many don’t — they don’t graduate to our list of approved products.

These tenets include strong alerting and reporting so we have clear visibility into the health and performance of the application software as well as the infrastructure on which they run. We make sure that something as simple as a bad log file doesn’t cause threads to back up and that failure points (database, memory capacity, etc.) have been reviewed and thoroughly tested.

Three years ago, as part of a continuing effort to improve system resiliency via greater simplification, we implemented a single set of “cookie-cutter” standards across the company — SOA middleware, blade servers, database and operating system. By the end of the year, the vast majority of our applications will run on our single cookie-cutter standard. This is a significant accomplishment in such a short time.

With this level of interoperable uniformity in place, the next problem to solve is the over isolation of applications on dedicated server farms that have either sprung up independently or were inherited via acquisition. Getting these systems onto a standard foundation enables us to benefit from the types of sharing that the mainframes offer so well, but at a price and level of flexibility they traditionally do not.

Mainframes are still the undisputed, seasoned masters of dependability. If you want something you can count on to run multiple types of services at very high volumes, they are the standard by which all others are measured. Open systems will catch up eventually, but they aren’t there yet.

Of course, from a simple unit cost standpoint, every time Moore’s Law does its thing, the number of calculations you get for your dollar increases. This is incredibly significant to the future of

computing. I once read on FutureTimeline.net that in the year 2000, a thousand dollars would have bought you compute power capable of processing the same number of calculations per second that an insect's brain processes. This year, that thousand dollars will buy you compute power equivalent to a mouse's brain. That's a pretty impressive increase in such a short time.

By 2038, your thousand-dollar computer will be comparable to your own brain in terms of how quickly it will think — and if that's not impressive enough (or scary enough depending on how you look at it), in 2060, your thousand-dollar computer will have enough compute power to go toe-to-toe with the combined brainpower of every human being on the planet. I'll be dead by then and, between you and me, I think that's OK.

None of this is true for mainframe computers — at least not as they are currently priced. And, as a result, the unit-cost gap between the technologies will get wider every year. Now I say "unit cost" because it's important to stress the point that the cost of the hardware isn't all you pay for with open systems. With all of that equipment, there is simply a lot more to manage, maintain and operate. Today, those extra costs eat up much of the cost differences between the two environments, so that's where we are currently focusing our efforts.

In 2007, I was with two members of my team (let's call them Jim and Glenn) sitting in a bar in San Jose, California, postulating the maturity gap between mainframe and open systems and what might be done to bridge it.

One big advantage the mainframe has, we all agreed, is that its operating system has all of its resources laid out before it and can subsequently manage load where it best sees fit. Mainframe central processing units (CPUs) aren't dedicated to specific tasks and, accordingly, don't sit idle while other CPUs are gasping for help. Some mainframes — indeed some large Unix servers — even offer CPU capacity on demand. That is, you don't buy them until you decide that you need them.

Why couldn't we do the same thing with blade servers? Lots of blade servers! One damp, scribbled napkin and lots and lots of testing later, we have developed an autonomous private cloud concept we call Organic Server Management, or OSM.

The dynamic nature of OSM is the key to its importance because it finally allows us to gain consistent quality of operations by not only recording and retaining valuable experience, but also building on it in a process of "continuous improvement."

The workload of today's operators is undoubtedly more complex than when I was a lad, following and automating simple, rhetorical scripts, but the basic concepts are still the same. They observe an "action" and then determine the appropriate "reaction."

The more skilled and experienced the operators, the better they can recognize various patterns as they emerge, such as observing the fact that a set of services is slowing down could suggest a number of possible conditions to a novice operator whereas the more seasoned among them may know that this particular service

HIGHLIGHT

We strictly enforce well-proven architectural tenets in all of our systems — tenets that improve the resiliency of our software.

has an occasional tendency to run out of available threads at this time and needs to have them cleared.

The operator may notice that a certain service's garbage collection pause times are starting to climb and knows to alert the development team and that he should relieve the situation by recycling the server — after first ensuring that it has completed processing its current workload and no more is sent to it.

The decision processes that operators (or in fact any of us) go through are mentally assembled each time they observe a condition and learn how to respond to it. More often than not, the more experienced the operator, the more likely he is going to know what to do. And then he leaves the company ... either to finish off his days gardening or to get a job at one of our competitors.

Then the process starts all over again with more-junior, less-skilled personnel who see the same conditions but either don't have the experience to know what to do or forget this particular situation because it's not that common or maybe they're just having an off day. It happens. "To err is human," said the human, rather

apologetically, I imagine. So maybe we need something else, something better.

OSM comes with a policy engine that allows us to automate repeatable behaviors — just like we do millions of times a day in the application software that powers our business. We've designed it to evolve organically to introduce itself to its human counterparts gradually, starting off with suggestions to simple conditions, such as: "The Web servers are running at 70 percent capacity, and we are about to enter our peak period. Based on historical data, which I can show you if you would like, I recommend that we provision an extra 50 percent capacity. Click 'yes' if you would like me to do this for you." Or maybe something a little less wordy.

If the operator disagrees with the recommendation, he or she will select "no," and the designers and tuners of OSM will work to understand what correction needs to be made so eventually they will say "yes." At which time, OSM will dynamically provision the servers with the appropriate software (OS, DMBS, COTS, OSS, applications, etc.), storage and network capabilities, which then come online to begin their day.

After the recommendations have been successfully accepted a sufficient number of successive times, the steps will be automated and the message changed to read: "Web server farm hit 70 percent capacity at 09:02:18, 50 percent extra capacity is currently being added. Click 'Cancel' to abort or 'Suspend' to adjust."

Then, slightly more-complex conditions will be tackled: "Pricing Server 14 has stalled. Developer team has been notified, recommend recycle. Click 'yes' to continue." And so on.

As more and more conditions are automated, the opportunity for human error is diminished. Stability improves and, along with it, so does our ability to more efficiently use our resources, both technical and human.

Our mainframe costs are already being surpassed by our open-systems costs, simply because that's where our development and transactional growth resides. More content, more shopping options, more features equals more servers and, potentially, more complexity.

Organic Server Management, and solutions like it, are going to revolutionize the world of midrange, open-systems operations as much as they are going to rationalize and bring order to it so we can continue to focus more and more of our resources on those aspects of our business that differentiate each of us from our competition — and fewer resources on those aspects that don't. ■