THE DATA-CENTRIC FUTURE

Part 1: Vision

by Stan Schneider, Ph.D.
CEO and Founder, Real-Time Innovations

JULY 18, 2006

The new opportunity for pervasive data.

Truly profound technologies become part of everyday life. Motors, plastics, computers, and now networking have made this transition in the last 100 years. These technologies are embedded in billions of devices; they have melted into the assumed background of the modern world.

Another step is emerging in this progression: pervasive, real-time data. This differs from the Internet in that this pervasive information infrastructure will connect devices, not people. Just as the "web" connected people and fundamentally changed how we all interact, pervasive data will connect devices and change how they interact.

Today's network makes it easy to connect nodes, but not easy to find and access the information resident in networks of connected nodes. This is changing; we will soon assume the ability to pool information from many distributed sources, and access it at rates meaningful to physical processes. Many label this new capability the "network centric" architecture. We prefer the term "data centric" because the change, fundamentally, is driven by a fast and easy availability of information, not the connectivity of the network itself. Whatever the name, it will drive the development of vast, distributed, information-critical applications.

Real-time middleware is the technological key driving the data-centric transformation. This three-part whitepaper offers a high-level, practical view of the present and future of distributed data transport, storage, and management. Part 1 starts with an overview of the state of the art of real-time networking middleware, and develops a vision for the pervasive-information future. Part 2 focuses on the challenges of implementing this vision, notably examining the performance and scalability issues in providing pervasive, real-time data. Throughout, we pose the toughest questions facing high-performance distributed systems today, and try to develop meaningful answers.

Why is there a new opportunity for pervasive data?

The technology is maturing that allows a network to support data-critical distributed applications. There are three important fronts: the rise of the data-centric thought model, emerging embedded standards, and nascent techniques for integrating embedded systems with the enterprise. We examine the first two here.

Data-centric design and publish-subscribe

The real change is from code-centric or architecture-centric thinking to data-centric design. Instead of configuring clients and servers, or building data-access objects and invoking remote methods, data-centric design implies that the developer directly controls information exchange. Data-centric developers don't write or specify code. They build a "data dictionary" that defines who needs what data. Then they answer the information questions: How fast is the data coming? Does it need to be reliable? Do I need to store it? Where is it coming from? What happens if a node fails?

With this information in hand, the next task is to map the information flow. Publish-subscribe middleware is the key enabling technology for data-centric design. Publish-subscribe nodes simply "subscribe" to data they need and "publish" information they produce. Thus, the information flow map is directly translatable to publishers and subscribers. The middleware does the rest: messages pass directly between the communicating nodes. The fundamental communications model implies both discovery—what data should be sent where—and delivery—when and how to send it.

This design should be familiar; it mirrors time-critical information-delivery systems in everyday life. All mass-media communications, including television, radio, magazines, and newspapers, are fundamentally publish-subscribe technologies. Publish-subscribe systems are good at distributing large quantities of time-critical information quickly, even in the presence of unreliable delivery mechanisms.

With direct access to data, system integration is orders-of-magnitude easier. Designers have always built interface specifications that detail which information flows between system components. With a publish-subscribe information bus, the interface is the design; interface specifications can essentially be directly implemented. This eliminates numerous intermediate steps, and with them all the overhead and opportunity for error they entrain.

The emergence of DDS: the embedded publish-subscribe middleware standard

In the embedded space, hundreds of designs have now been completed with advanced publish-subscribe middleware. This experience drove the first accepted standard that targets real-time distributed systems, the Object Management Group's Data Distribution for Real Time Systems Standard (known as "DDS").

DDS is sophisticated technology; see [DDSSpec] for technical details. It goes well beyond simple publishing and subscribing. However, it is based on very familiar concepts. We present a high-level analogy here.

DDS is like a neighborhood news service for computers. It puts reporters at every "house" and delivers the stories they write to every other house that cares to subscribe.

Each house has fine control over what news it gets when. You can subscribe to every single update on every topic or weekly summaries of just one. You can have reliable delivery directly to your iPod, or have them just toss a paper on your doorstep when they can. You don't know or care where the stories come from, so multiple reporters can back each other up to make sure you always get your morning traffic report. You can even set deadlines for delivery, ask to be notified when your neighbors are on vacation, and much more.

Anyone who has participated in a grade-school carpool can appreciate the utility of such a system. Instead of coordinating complex schedules and the incessant changes, you could just subscribe to your carpool group's status updates. You would immediately know when Cindy's mom left her house, be informed that Billy is sick today, and know that Ryan has early swim practice, all without making a phone call. In the more general case, this information could also be available in your car so you (or the car itself) can re-plan en route…through the power of real-time pervasive information.

Carpooling is but one of many applications that would be transformed by this technology. Pervasive information will someday transform everyday life.

Of course, this is just an analogy; DDS does not target neighborhood or inter-human communications. DDS targets high-performance networks. It can deliver 10,000 "stories" a second to any or every "house" on your network. The "stories" are data updates of any type; "houses" are any computer or embedded device. The "fine control" options are Quality of Service (QoS) parameters that fine-tune delivery.

This infrastructure fundamentally changes how easy it is to set up a highly-connected networked application. Distributed, information-critical applications are the next generation of computing. The impact on complex real-world applications will come much sooner and be just as transformational.

Before we get too deeply into this issue, let's examine a more fundamental question: if middleware is the key technology enabling data exchange, what is middleware?

What is middleware, anyway?

"Middleware is everywhere." With this recent advertising campaign, IBM raised the global level of awareness of middleware. The ads, however, did little to resolve the more cogent question: what is middleware?

The broadest definition of middleware is simple: middleware is any layer of software between the application and the operating system. The application is the specific-purpose "top level" software; examples range from web browsers to enterprise banking systems. The operating system (OS) is the lowest-level code that controls the computer itself; the OS controls the various devices and peripherals, schedules tasks, and implements a basic network stack. Middleware is everything else.

This definition is simple, but it's not particularly useful. The gap between the application and the OS is huge; it arguably includes most software, including network protocols, databases, graphics libraries, web servers, document management systems, and much, much more. Complex systems are developed in layers. By the broad definition, most any layer, once it's sufficiently mature to be reused in a general role, can be called middleware.

Networking Middleware

There is a more useful definition. Increasingly, with the rise of networking, the term "middleware" is becoming synonymous with "networking middleware": layers above the basic TCP/IP stack that implement sophisticated data transfer and management models. Networking middleware consists of software agents that shuttle information between different components of complex, distributed applications.

Even within this narrower range, there are many types of middleware. Distributed Hash Tables and peer-to-peer document sharing, including technologies like BitTorrent and Kazaa, create a distributed source for individual files. They were first widely used (or abused) by music sharing services. These technologies are massively scalable sources of relatively low-bandwidth data. They make virtually no attempt to keep the data consistent throughout the network.

Distributed databases provide much higher bandwidth and data consistency than peer-to-peer sharing systems. In contrast to the above, they often strive for very high-fidelity data control. In fact, transactional systems, designed for applications like banking, offer guaranteed integrity and persistent data, even in the face of system failures. However, high-performance scalability is a challenge.

Data distribution and messaging systems strive to update multiple nodes with rapidly changing data at speeds measured in microseconds. Technologies include the traditional "middleware" such as CORBA, message queues, and JMS, as well as publish-subscribe newcomers like DDS. While some of these systems boast impressive data transfer performance, most target smaller systems of only a few hundred nodes, and leave data consistency as an exercise for the application.

Challenge for the Future

The really interesting development today is the merging of all these technologies. These combinations bring vast new capabilities; they also impose tradeoffs. The most important tradeoffs are between performance, scalability, and data consistency. For instance, data distribution can be used to greatly extend the reach of a database system, but it makes high-integrity transactions a challenge. The massive scalability achieved by peer-to-peer sharing systems isn't so easy when high performance is required.

Still, the power of combining data transport, storage, and discovery into a truly pervasive information management system is compelling. Any biologist will tell you that zones where environments mix support the most diverse life. Similar forces are driving a new ecosystem of data distribution today.

What is real time?

So, technologies to find, store, and deliver data are on the horizon. But information has always been available given enough time and resources. The power of pervasive data, however, is to have that data easily available in real time. Thus, we need to examine a more basic issue: what is real time?

Many authors have defined "real time" in many ways. One textbook definition: A computer system is real-time if a correct response must not only be computationally accurate, but also timely with respect to its external environment. So, a brake warning system for a train is real time if it not only indicates that the train should stop, but also does so in time for the train to avoid smashing into the school bus. An answer that's even a little late is wrong.

Many systems can withstand a few late answers, as long as most aren't late, and the few that are late aren't too late. This is often called "soft" real time. For a soft real-time system, being on time is defined stochastically.

So, there really is no crisp definition of real time. There's a wide range, from mathematically-provable response times to "fast enough". In practice, two general meanings have arisen for real time, one in enterprise software and one in embedded systems.

In the enterprise, a system is real time if it responds "now" as perceived by a human. Thus, a system that reports stock trades within a few minutes is real time; the stock listings in yesterday's paper are not. A "real time" airline reservation system that takes 10 seconds for each request is likely acceptable. One that takes 10 minutes is not. The challenge in being real time is often accessing or searching through large databases of information and presenting them in an intuitive display.

In embedded systems, real time means predictable and fast with respect to the relevant physical processes. Generally, an embedded system that reliably responds in a millisecond or so is considered real time. The primary challenges are locating the right information, determining where it should go, and delivering it quickly.

So, real time in the enterprise is orders of magnitude different than real time for embedded applications. There's little value in exploring this further here. Suffice it to note that in both domains, real time is more about predictable reliability than it is about performance. Different systems have very different requirements for time determinism and responsiveness. Meeting these requirements is the key challenge of real-time system architecture.

What makes a system real time?

Whatever the definition, how do you make a system real-time?

There is a famous Stanford PhD qualifying-exam question that asks, "Why are clear things clear?" It's a trick question, because there is no property of materials that yields clearness. Things are clear because they don't absorb, reflect, or scatter light. Fundamentally, they are clear because they fail to be unclear.

Systems are real time for a similar reason: they fail to be non-real time. Consider a control system that must respond to a sensor (e.g. a temperature rise) by changing some actuator (e.g., closing a valve). To respond to the stimulus, the sensor must detect the stimulus, that detection must be reported to peripheral hardware, the peripheral must interrupt the CPU, the operating system must process the interrupt, the application must calculate a response, the response must be passed to the actuator, and the valve must close. If each of these steps takes fixed time, the system will be deterministic. If those times sum to a small enough value, the valve may close before the boiler overheats, and the system can be called real time. If any of these steps takes an unknown or excessive time, the system will fail to be real time.

If the sensor and actuator are connected over a network, the chain of events minimally includes creating a network message, passing that message through the network stack, putting the message through some transmission medium, receiving the message at the other node, passing it back up through the stack, and interrupting another processor that controls the valve. Even this simple one-to-one system has many more opportunities to fail to be real time.

What makes middleware real time?

Complex, distributed real-time systems present the interesting new challenge. Throwing networking into the mix complicates things greatly. Network hardware, software, transport properties, congestion, and configuration effect response time. "On time" may have different meanings for different nodes. Networks merge embedded and enterprise systems, combining the challenges facing both. Even the simplest question of all, "what time is it?" is hard to answer in a distributed system.

But although it may be tough, solving the networked real-time challenge is critical. It is the key to pervasive data.

So, what makes middleware real time? The easy answer would be that middleware is real time if it can always process requests in a sufficiently-short deterministic timeframe1. Unfortunately, this is rarely possible or even definable. Most designers cannot assume a reliable or strictly time-deterministic underlying network transport, as most real-time systems must operate without these luxuries. Most architects do not know the real performance bounds of their systems. Usually, you know only that the network transport affords the raw ability to succeed if properly managed. The key to successful distributed real-time middleware is to optimize those last two words: "properly managed".

Pervasive data requires two things: data availability and real-time performance. We have discussed that real time means fast enough to reliably effect the environment. Part 2 will focus on this question: how do we achieve real-time?

[1] Of course, none of the rest of the system can fail to be real-time. Thus, for instance, for maximum determinism, you need a real time operating system and a sufficiently fast, unloaded network. Other sources have covered these issues in depth.

What does the future hold?

There is a new generation of connected systems in sight. Data-centric architectures will change the world by making information truly pervasive. Pervasive information, available at nearly-instant speeds, will enable much more capable distributed, data-critical applications.

Real-time middleware is the technological key driving the data-centric transformation. The next exciting step is merging information distribution, storage, and discovery into a pervasive data model. These technologies are evolving rapidly. They will soon satisfy and combine the real-time performance requirements of embedded systems and high-performance data access needs of the enterprise. Although there are many challenges in performance, scalability and data integrity, a data-centric, pervasive-information future is coming. Soon.