Data-centric pervasive information is the wave of the future

This article explores the evolution from data networks to pervasive data. With pervasive data, all information is available anytime at any place, without consideration of its origin. The article offers a high-level, practical view of the state of distributed data transport, storage and management. It starts with the basic state-of-the-art of real-time networking middleware, addresses the tough performance issues, and finally develops a vision for the pervasive-information future.

By Stan Schneider

The coming information infrastructure for pervasive, real-time data differs from the Internet. This new data-centric network will connect devices, not people, and will change how devices interact. The technological key driving this data-centric transformation is real-time middleware, but to fully exploit the opportunities of data-centric networking there needs to be a change in the thought model that designers apply.

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. This "data-centric" architecture will drive the development of vast, distributed, information-critical applications.

The change must begin with a shift from code-centric or architecture-centric thinking to data-centric design, which is a fundamentally different approach. 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 information questions about the data's source, how fast the data is coming, its need to be reliable, its need for storage, and what happens if a node fails.

With this information in hand, the developer then maps the information flow using the publish-subscribe information flow model. This model should be familiar; it mirrors time-critical information-delivery systems used in everyday life, including television, radio, magazines and newspapers. Such publish-subscribe systems are good at distributing large quantities of time-critical information quickly, even in the presence of unreliable delivery mechanisms.

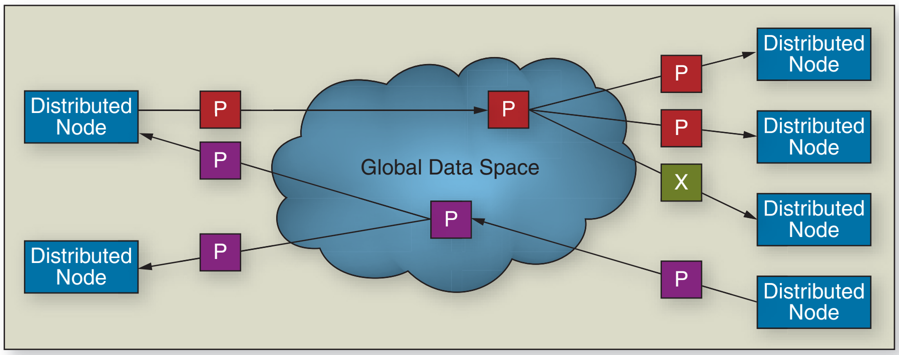

Figure 1. The publish-subscribe model creates a virtual global data space that nodes can read from and write to in order to exchange information.

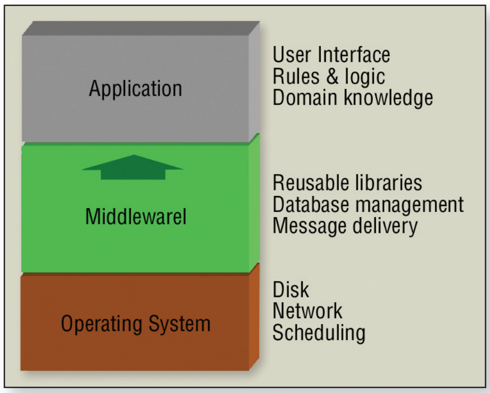

Figure 2. In the data-centric model, middleware handles the communications of data among applications, freeing the developer from dealing with the details of data transport.

Publish-subscribe works by creating the illusion of a shared global data space within the network (Figure 1). This data space contains data objects that applications in nodes distributed across the network can access using simple read and write operations. A node that writes data to the space is said to "publish" the data and a node that reads data is a "subscriber."

To use this data flow model, applications must be programmed in a new way. Rather than simply asking for data from each other as needed, the application must separate the information-access intent (what it wants to do) from the information exchange itself. This means that the application must declare its intent of writing data and specify which data objects it will write (i.e., define its publications). It must also declare its intent to read data and specify which data objects it intends to read (i.e., define its subscriptions). Both declarations must be made before the application actually writes or reads the data itself.

By thus separating intent from exchange, the application provides the middleware with advanced information about data exchanges, giving the middleware an opportunity to reserve resources so that the information access can be as efficient as possible. The information flow map that the publish-subscribe declarations describe thus directly translates to a set of logical communications channels between publisher nodes (data sources) and subscriber nodes (users of data). The developer does not need to implement these channels because publish-subscribe middleware on the network handles the actual communications. The middleware passes messages directly between the communicating nodes and handles both discovery—what data should be sent where—and delivery—when and how to send it.

Middleware a key enabler

Publish-subscribe middleware is the key enabling technology for data-centric design.

The DDS standard

The data distribution service (DDS) specification standardizes the software application-programming interface (API) by which a distributed application can use "data-centric publish-subscribe" (DCPS) as a communication mechanism. Since DDS is implemented as an "infrastructure" solution, it can be added as the communication interface for any software application.

Advantages of DDS:

- based on a simple "publish-subscribe" communication paradigm;
- flexible and adaptable architecture that supports "auto-discovery" of new or stale endpoint applications;
- low overhead—can be used with high-performance systems;
- deterministic data delivery;
- dynamically scalable, efficient use of transport bandwidth;
- supports one-to-one, one-to-many, many-to one, and many-to-many communications; and
- large number of configuration parameters that give developers complete control of each message in the system.

DDS provides an infrastructure layer that enables many different types of applications to communicate with each other. The DDS specification is governed by the Object Management Group (OMG), which is the same organization that governs the specifications for CORBA, UML and other standards. A copy of the DDS specification can be obtained from the OMG website at www.omg.org.

For one thing, it makes system integration easier. Designers have always built interface specifications that detail which information flows between system components and created the buses that carry that information. Changes to the information content can have a ripple effect throughout the system, requiring changes to many interrelated components. With a publish-subscribe network, the information flow specification is the design. The interface specifications are, essentially, directly implemented by the middleware. Individual nodes simply "subscribe" to data they need and "publish" information they produce and the middleware does the rest.

The middleware, also called "networking middleware," is composed of software layers above the basic TCP/IP stack that implement sophisticated data transfer and management models (Figure 2). These layers implement software agents that shuttle information between different components of complex, distributed applications. There are many types of such middleware besides the publish-subscribe model, such as document sharing, distributed databases, and messaging systems, but most fall short of creating a pervasive, real-time data network.

Distributed Hash Tables and peer-to-peer document sharing—technologies like BitTorrent and Kazaa—create a distributed source for individual files. These technologies are massively scalable sources of relatively low-bandwidth data. They make virtually no attempt, however, to keep the data consistent throughout the network.

Distributed databases provide much higher bandwidth and data consistency than peer-to-peer sharing systems. In addition, 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. High-performance scalability, however, is a challenge for such systems. The random-access semantics of memory and the implied totally reliable "instantaneous" response cannot be implemented transparently in a network where computers can join and leave and communications links have sporadic faults.

Data distribution and messaging systems, which includes the publish-subscribe model, 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. 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.

DDS enables the net-centric vision

The need for data-distribution is fundamental to C4I as well as other domains such as traffic monitoring and industrial sensing.

Within the GIG there are complex dataflows (one-to-many, many-to-one, many-to-many) with differing requirements in terms of real-time performance (update rates, latency, bandwidth); reliability; fault-tolerance (redundancy, automatic failover, no single point of failures); dynamic configuration changes (participants may join or leave arbitrarily); and scalability. The GIG environment reflects the fluid and ever-changing nature of the real world.

The lack of established standards has led to the development of ad-hoc custom one-off solutions with sub-optimal performance and robustness that end up being expensive to maintain. Commercial implementations of publish-subscribe data-distribution middleware have existed and have been deployed with success. However, each vendor provided a different API and slightly different semantics for equivalent concepts.

This situation was similar to the one prior to the adoption of UML. Successful modeling tools existed but widespread acceptance,

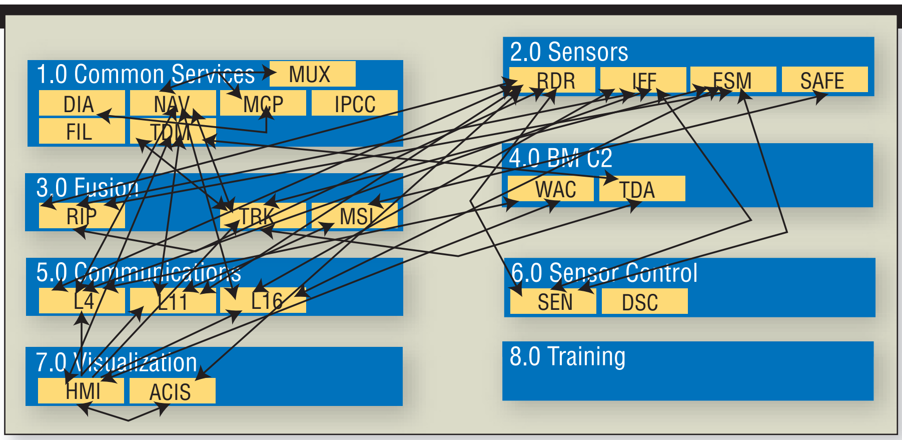

Figure 3. The systems of the Hawkeye aircraft contain many communicating modules, forming a complex data flow pattern that can be hard to integrate and maintain.

availability of trained developers, and broad knowledge exchange did not come until the UML standard consolidated terminology and notation. The recent adoption of the Data-Distribution Service (DDS) for real-time systems specification by the Object Management Group (OMG) is doing for data-distribution what UML did for modeling technologies. (See sidebar, "The DDS Standard.")

The DDS is a middleware technology that is especially well suited to address the needs of the C4I applications within the GIG. DDS uses a publish-subscribe approach to let application developers and system integrators focus on the logic of the applications and communicate by publishing the information they have and subscribing to the information they need. DDS handles automatic discovery, reliability and redundancy over otherwise unreliable networks. A key aspect of the DDS standard is the pervasive use of quality of service (QoS) to configure the system and establish middleware-brokered QoS contracts between publishers and subscribers. QoS contracts provide the performance predictability and resource control required by real-time C4I systems while preserving the modularity, scalability and robustness inherent to the anonymous publish-subscribe model.

DDS is well suited for heterogeneous networks as it handles format conversion across operating systems, processor architectures and programming languages.

These characteristics make it ideally suited for the heterogeneous dynamic environments introduced by the increased use of wireless devices within the GIG.

The QoS parameters help configure the system to ensure that each application receives its data in what the application considers to be a timely fashion. Publishers define a maximal level for each QoS policy, declaring their best ability to deliver data. Subscribers request a minimum level for each QoS policy. The middleware then brokers a "contract" between the two and reserves the resources needed to support the data exchange at the right QoS level. These contracts provide the performance predictability and resource control required by real-time and embedded systems. At the same time they help preserve the modularity, scalability and robustness inherent to the anonymous publish-subscribe model.

These contracts describe a one-to-one data exchange but can do more. Special QoS parameters also support such things as event propagation and messaging. This allows applications to publish data that has many subscribers, creating a broadcast effect for data that is not critical, such as a series of frequent temperature readings that control systems are monitoring. In such applications, the loss of an occasional reading has no impact. Using the broadcast approach, then, reduces traffic on the network by eliminating the need for acknowledgments and re-sends of data.

Information-critical applications simplified

These flexible data delivery mechanisms along with the speed and reliability of data transfers that DDS provides enable the creation of distributed, information-critical applications that are the next generation of computing. A real-world example can be taken from the design of the Navy's E-2C Hawkeye aircraft. The various software modules in the system communicated information to one another using direct links, quickly resulting in a complicated information infrastructure (Figure 3) that was difficult to integrate. Changes to one part of the system could affect many other parts (Figure 4). The data flow, data-centric thinking transformed this design by replacing all of the individual links with a publish-subscribe structure (Figure 5), greatly simplifying system integration and maintenance.

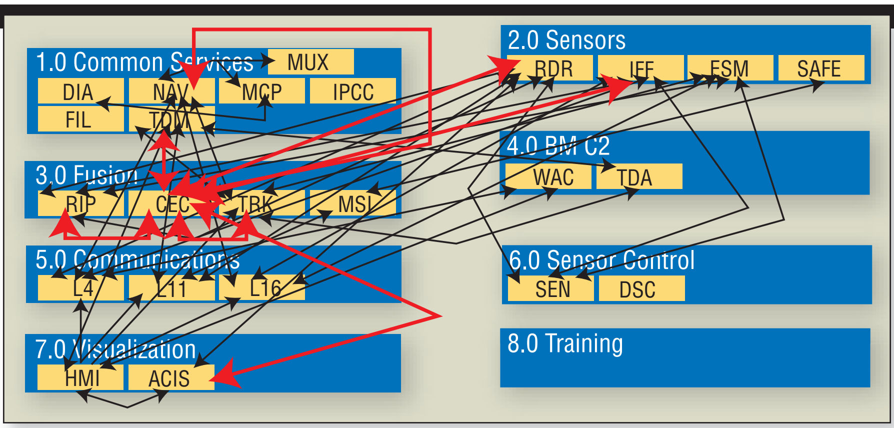

Figure 4. Complex data flow patterns can result in equally complex interactions where a change in one module forces changes (shown in red) in many others in order to maintain a match among the communications links.

The simplified distribution of data is only one element of the power inherent in DDS, however. Technologies to find, store and deliver data have always been available if there is enough time and resources. The situation becomes much more interesting, however, if the data is easily available in real time. There really is no crisp definition of "real time," however. The term can refer to anything from mathematically provable response times to "fast enough." In practice, however, 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 to achieving real-time performance in enterprise systems typically involves accessing or searching through large databases of information and presenting them in an intuitive display.

In embedded systems, real time means being predictable and fast with respect to the relevant physical processes. The primary challenges here are locating the right information, determining where it should go, and delivering it quickly. Consider a control system that must respond to a sensor (e.g., a temperature rise)

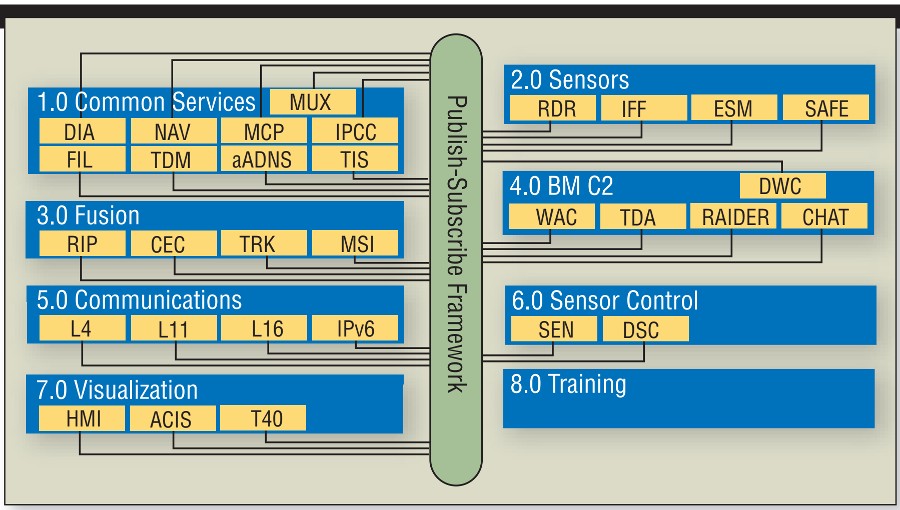

Figure 5. The publish-subscribe data model simplifies the interactions among software modules, replacing point-to-point channels with communications to and from a framework that handles the data distribution.

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 add up to a small enough value, the valve can close before the boiler overheats and the system can be called real time. Generally, an embedded system that reliably responds in a millisecond or so is considered real time.

Making middleware real-time

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 time frame. Unfortunately, this is rarely possible or even definable. Most designers cannot assume a reliable or strictly time-deterministic underlying network transport, so most real-time systems must operate without these luxuries and simply concentrate on getting the information to its destination on time.

Making the system distributed across a network greatly complicates things. Network hardware, software, transport properties, congestion and configuration affect the system's response time. Furthermore, the definition of "on time" may have different meanings for different nodes in a distributed system. Systems that merge embedded and enterprise applications are even more complicated, combining the challenges facing both.

Fortunately, strict time-determinism in the network may be unnecessary. Most architects do not know the real performance bounds of their systems. They know, however, that their network transport affords the raw ability to succeed if properly managed. The key to successful distributed real-time middleware, then, is to "properly manage" the delivery of data.

The publish-subscribe model makes such management possible. The many logical pathways, redundant data sources, and timely distribution of data that the model supports provide the tools that applications developers need in order to handle the unreliability and lack of determinism in the underlying network. The result can be an information-handling system that pools the resources of many distributed data sources and delivers it at rates that are applicable for the control of physical processes.

Many challenges still need to be addressed in performance, scalability and data integrity, but a data-centric, pervasive-information future is coming. The technologies for information distribution, storage and discovery are evolving rapidly and are merging into a pervasive data model. They will soon satisfy and combine the real-time performance requirements of embedded systems and high-performance data access needs of the enterprise. Data-centric architectures such as DDS will change the world by making information truly pervasive. That pervasive information, available at nearly instant speeds, will enable much more capable distributed, data-critical applications. **DE**

ABOUT THE AUTHOR

Stan Schneider is chief executive officer of Real-Time Innovations Inc., which he founded in 1991 to develop productivity tools for real-time applications. A recognized expert in real-time software systems and architectures, Schneider holds a BS in Applied Mathematics (Summa Cum Laude) and a PhD in Electrical Engineering and Computer Science from Stanford University. He can be reached at Stan.Schneider@rti.com.