A DATA-ORIENTED SOFTWARE ARCHITECTURE FOR TELEMETRY

RAJIVE JOSHI, Ph.D.
Real-Time Innovations, Inc.
385 Moffett Park Drive, Sunnyvale, CA 94089
+408.990.7400
rajive@rti.com

ABSTRACT

Building modern telemetry systems is fraught with challenges involving subsystem integration, the role and management of data, scalability issues, disparate technologies, concerns about cost-effectiveness and more. This article addresses today's challenges with a solution based on adopting a data-oriented architecture and relying on a standards-based, integrated high-performance middleware platform with standards-based programmable components. Key to the solution is integrating around the system information model instead of the application or technology infrastructure. A standards-based middleware infrastructure that breaks away from traditional assumptions is at the core of this approach. The article also presents successful applications of data-oriented architecture using standards-based middleware.

1 INTRODUCTION

Building modern telemetry systems requires developing and seamlessly integrating various subsystems, including articles in space, air or ground or under water; a variety of links with differing characteristics; and disparate ground systems. Data is continuously acquired, transmitted, stored and analyzed in real time, and it must be made available to external systems. Besides the need to scale with an ever-increasing volume of data, there is a constant drain on the software infrastructure to accommodate new and legacy article, link and ground-system technologies easily and cost effectively.

Today a solution is available that successfully addresses each of these challenges. It is based on adopting a data-oriented architecture, which results in loosely coupled components. This solution relies on a standards-based, integrated high-performance middleware platform that provides standards-based programmable components for messaging and data distribution, storage and data management, complex event processing for real-time analysis and decision making, visualization, fault tolerance, recording and retrieval, security, and technology adapters. Such a solution is integrated around the system information model instead of the application or technology infrastructure of choice.

At the core of the solution is a standards-based middleware infrastructure that breaks away from traditional assumptions. This approach resolves the following issues effectively: single points of

failure or loading, unreliable transports, changing topologies, delays related to local performance and the need for system adaptability.

2 TELEMETRY SOFTWARE ARCHITECTURES

Modern telemetry systems are complex systems facing complex missions (see Figure 1). Today's software designs are effective but lack the flexibility required to adapt quickly to multiple uses. Most remote sensing units require a special-purpose monitoring station, which is well integrated but expensive. Remote-sensing software is typically redesigned for each new sensing unit or article and even sometimes for different payloads on the same unit. Several data links are used, and standards are emerging to address the need for monitoring ground stations to operate multiple remote articles. However, no clear, standard way to integrate the links with either end has emerged. In addition, the monitoring ground stations must make the data available to a variety of external systems for storage, archiving, indexing and interpretation.

Telemetry architectures today approach the integration of remote articles, links and ground stations at an application level. The handshake protocol or the logic to request missed or historic information is in the application code. This close coupling makes it difficult to make modifications to existing system components. Underneath the cover, many of these distributed systems can be broken into a collection of hard-wired, static *point-to-point* communication links. Point-to-point systems have several disadvantages (such as being difficult to modify and offering limited scalability) and result in an overall brittle architecture.

Figure 1 Building modern telemetry systems requires developing and seamlessly integrating various subsystems, including articles in space, air or ground or under water; a variety of links with differing characteristics; and disparate ground systems.

The data-communication links between the remote articles and the ground stations are constrained in nature. The links may be transient, bandwidth-limited and/or lossy, and they often introduce long delays. In addition, multiple paths and links may occasionally be available.

Coupling the link characteristics and behavior into the remote article-sensing software and the ground-station software is expensive and hard to maintain.

Furthermore, ground stations have to deal with the challenges of interoperability with other systems and must effectively address data management and security using industry standards and open architecture to ensure longevity and expandability.

In such a demanding environment, how can we deliver seamless **end-to-end** connectivity while optimizing end-to-end performance? In particular, it is important to strike the right balance of (1) **performance**, so that the system can efficiently leverage the underlying hardware infrastructure; (2) **scalability**, so that the system will keep up with the increasing demands; (3) **availability**, so that there is no downtime; (4) **security**, so that data and systems are protected; (5) **interoperability**, so that new and legacy components can be seamlessly integrated; and (6) **affordability**, so that the operational or administrative costs and system lifecycle costs are minimized.

3 A STANDARDS-BASED, DATA-ORIENTED SOFTWARE ARCHITECTURE

We address these concerns by introducing the framework of data-oriented architecture. The key is to separate data from behaviour and treat data as the primary artifact around which independent, loosely coupled behaviours are organized. Changes to data or information drive the interactions between components. The result is loosely coupled software components with data-oriented interfaces that describe the data produced or consumed and the associated Quality of Service (QoS) offered or requested. Such components can be seamlessly integrated using a high-performance, open standards-based messaging and caching infrastructure such as the Data Distribution Service (DDS). DDS implementations are available from many vendors.

3.1 ADOPT A DATA-ORIENTED ARCHITECTURE

With a data-oriented approach, we explicitly formalize the data and metadata produced and consumed by a component or a system, then use a logical data bus to connect them. A data-oriented interface describes the common logically shared data model and the messages produced or consumed by a component as a result of changes to the data model, as well as the associated QoS offered or requested (see Figure 2).

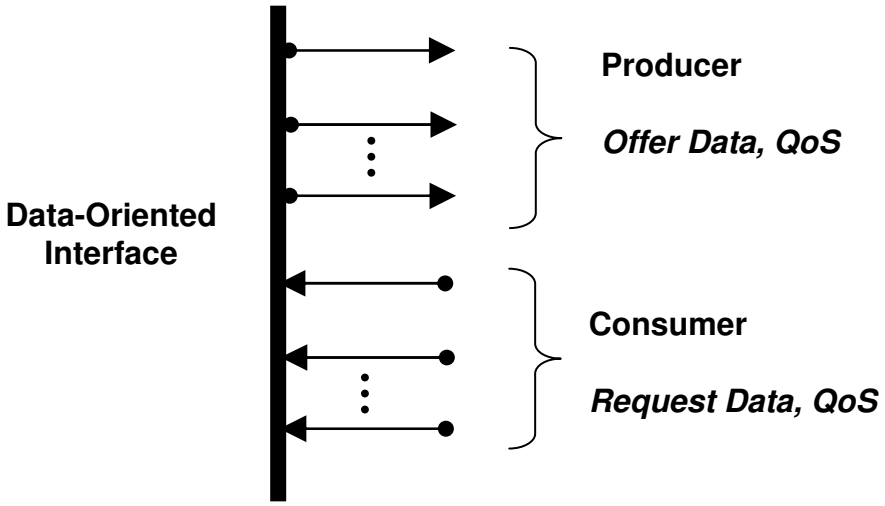

Figure 2 A data-oriented interface specifies (1) the data a component produces along with the offered QoS and (2) data a component consumes along with the requested QoS.

Working with data as a "first-class citizen" enables system designers to abstract the production, flow and consumption of data away from the details of processing that data. This shifts the focus to the overall system architecture and information flow.

The result is an information-driven, data-oriented architecture framework, where the changes to the interface data models drive the interactions between loosely coupled components. The responsibility for message routing and delivery and for managing QoS is delegated to the underlying messaging infrastructure (see Figure 3).

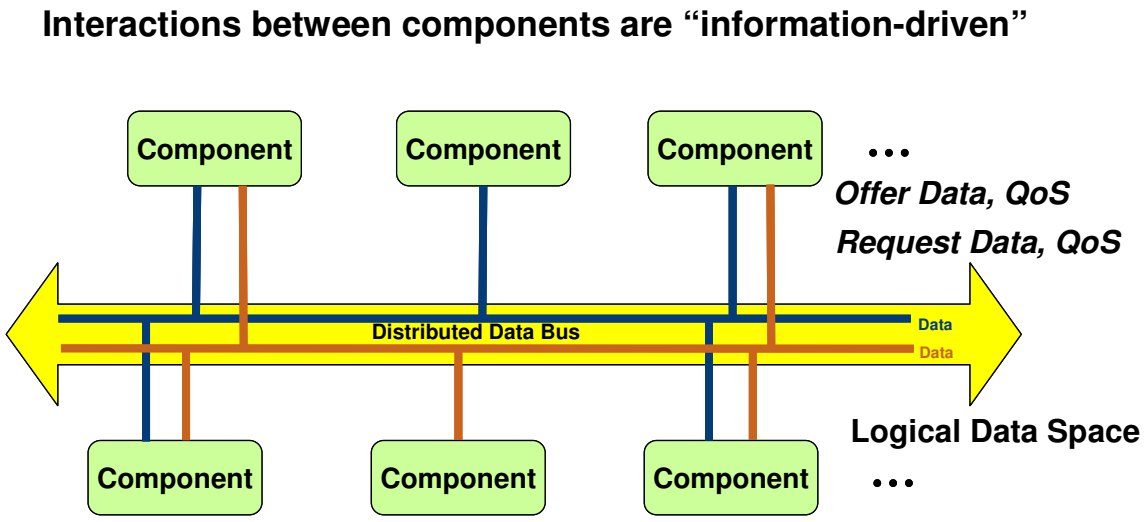

Figure 3 Components have a data-oriented interface for offering/requesting data and QoS, and changes to the data drive component interactions. An open standards-based distributed data bus infrastructure routes information and provides ready access to data in each component's local cache.

The traditional point-to-point approach of integrating components results in N*N point-to-point custom connections for each pair of components. The data-oriented architecture framework

reduces the integration problem from an O(N*N) problem to O(N). Components simply need to define and expose just the data model, and the underlying messaging infrastructure takes on the responsibility of delivering the right messages to the right components at the right time for components operating on same logical data model.

3.2 RELY ON A STANDARDS-BASED, INTEGRATED HIGH-PERFORMANCE PLATFORM

An appropriate messaging infrastructure for building loosely coupled systems using the principles of data-oriented programming is specified by the DDS open standard developed by the Object Management Group (OMG) industry consortium.

The result is a system software architecture where components communicate using data-oriented interfaces expressed in terms of standard DDS Application Programming Interfaces (APIs) and formally defined DDS QoS policies. These components are implemented using a standards-based DDS messaging protocol.

Several DDS implementations are available today. For example, the RTI Data Distribution Service provides a high-performance, constantly available, real-time distributed data bus. It does not rely on any servers or daemons, establishes peer-to-peer data paths between components and works over unreliable networks.

Leading DDS implementations such as RTI's provide the deterministic low-latency, high-throughput messaging and data caching important for real-time end-to-end performance. These implementations break away from the traditional messaging infrastructure assumptions in the following fundamental areas:

- **There can be no single point of failure or loading**. Traditional approaches use a daemon or a broker-based architecture, which can lead to partial failure or complex failure/repair modes during recovery. However, it is possible to implement the DDS specification as a peer-to-peer architecture that implements a decentralized messaging protocol. In such an implementation, there is no single point of failure or loading, or potential for priority inversion.

- **Transports are unreliable.** Networking transports drop packets, links go down and hardware fails. The reliability model, on the other hand, is built into the standard DDS messaging wire-protocol above the network stack. APIs allow applications to control the reliability and receive notifications of network errors.

  It is worth noting that the DDS messaging protocol is de-coupled from the data-link/transport implementation details. Thus the transport can be optimized separately from the application-level QoS.

- **Topology can change on the fly.** For mission flexibility, components must be field replaceable and operational parameters must be changeable on live systems without

disruption. The DDS infrastructure provides automatic self-discovery and configuration, so components can be added and removed dynamically on a live system. Nodes may also fail and restart at any time. Components can start in any order.

- **Local performance matters.** Transporting data from point A to point B is often not enough. The data may be needed later, without the penalty of retransmission delay. Either the application or the infrastructure thus must provide a local in-memory cache for all data. With DDS, providing the cache as part of the DDS infrastructure greatly eases data management.

- **Adaptable systems demand a tunable infrastructure.** Many types of information flow in telemetry systems, each with differing requirements. The requirements must evolve and adapt to new demands. The DDS infrastructure is tunable, so that applications can configure and optimize it to meet specific mission requirements.

The above attributes make DDS particularly well suited to the messaging requirements of telemetry systems.

3.2.1 THE DDS STANDARD

The OMG DDS standard is the first comprehensive specification available for publish-subscribe data-centric designs. [1] [2] [5]. It specifies both a wire protocol for interoperability and an API for portability.

As illustrated in Figure 4, DDS creates the illusion of a shared "global data space" populated by data objects that applications running on distributed nodes can access via simple read and write operations. In reality, the data does not really live in any one computer's address space. Rather, it lives in the local caches of all the applications that have an interest in it.

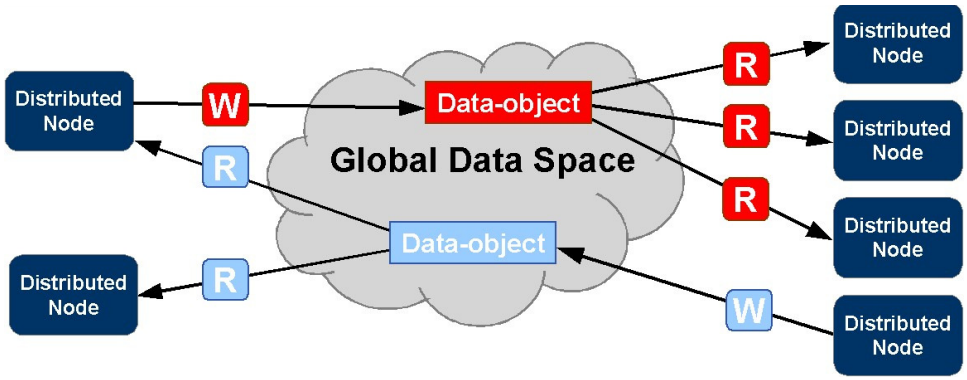

Figure 4 This overall DDS model depicts the ways distributed nodes are able to read (R) and write (W) data objects that live in a shared virtual global data space.

The DDS publish-subscribe model connects anonymous information producers (publishers) with information consumers (subscribers). The overall distributed application is composed of processes called "participants," each running in a separate address space and possibly on different computers. A participant may simultaneously publish and subscribe to typed data

streams identified by names called “Topics.” The interface allows publishers and subscribers to present type-safe API interfaces to the application.

DDS defines a communications relationship between publishers and subscribers. The communications are decoupled in space (nodes can be anywhere), time (delivery may be immediately after publication or later) and flow (delivery may be reliably made at a controlled bandwidth).

To increase scalability, topics may contain multiple independent data channels identified by keys. This allows nodes to subscribe to many, possibly thousands, of similar data streams with a single subscription. When the data arrives, the middleware can sort it by the key and deliver it for efficient processing.

DDS QoS parameters specify the degree of coupling between participants and properties of the overall model and of the topics themselves. Since DDS coupling is loose, communication paths can be defined, created and destroyed at run time. Further, implementations may automatically discover requesting publishers and subscribers, hooking them up at run time with no previous configuration.

DDS is fundamentally designed to work over unreliable transports, such as User Datagram Protocol (UDP) or wireless networks. No facilities require central servers or special nodes. Efficient, direct, peer-to-peer communications, including multicasting, can implement every part of the model.

DDS’s QoS configurability makes it well suited to lossy networks and transient links. It is proven in use for communication with production rail systems [7], as well as over intermittent satellite connections in test [8].

The extreme configurability offered by DDS makes it feasible to directly use the same protocol for data links as the middleware on the remote article and the ground station, making communications much simpler. In fact, with a direct DDS connection, either end could directly subscribe to any information produced by the remote component.

3.2.2 INTEGRATED END-TO-END PLATFORM

With the availability of a DDS backbone, it becomes possible to provide optimized application components for certain software categories to achieve an end-to-end middleware platform for high-performance real-time applications (see Figure 5). Application components are organized around an agreed-upon underlying semantic data model that is automatically mapped to the natural representation used by the application platform, in accordance with the principles of data-oriented design. Thus, a new application component can be rapidly developed by using off-the-shelf application platforms that have been a-priori integrated with and optimized for the messaging infrastructure. With DDS, commercially available out-of-the-box integrations of popular application platform components with open standards-based integrated messaging and caching middleware such as the RTI Data Distribution Service proceed more quickly and promote optimized end-to-end performance.

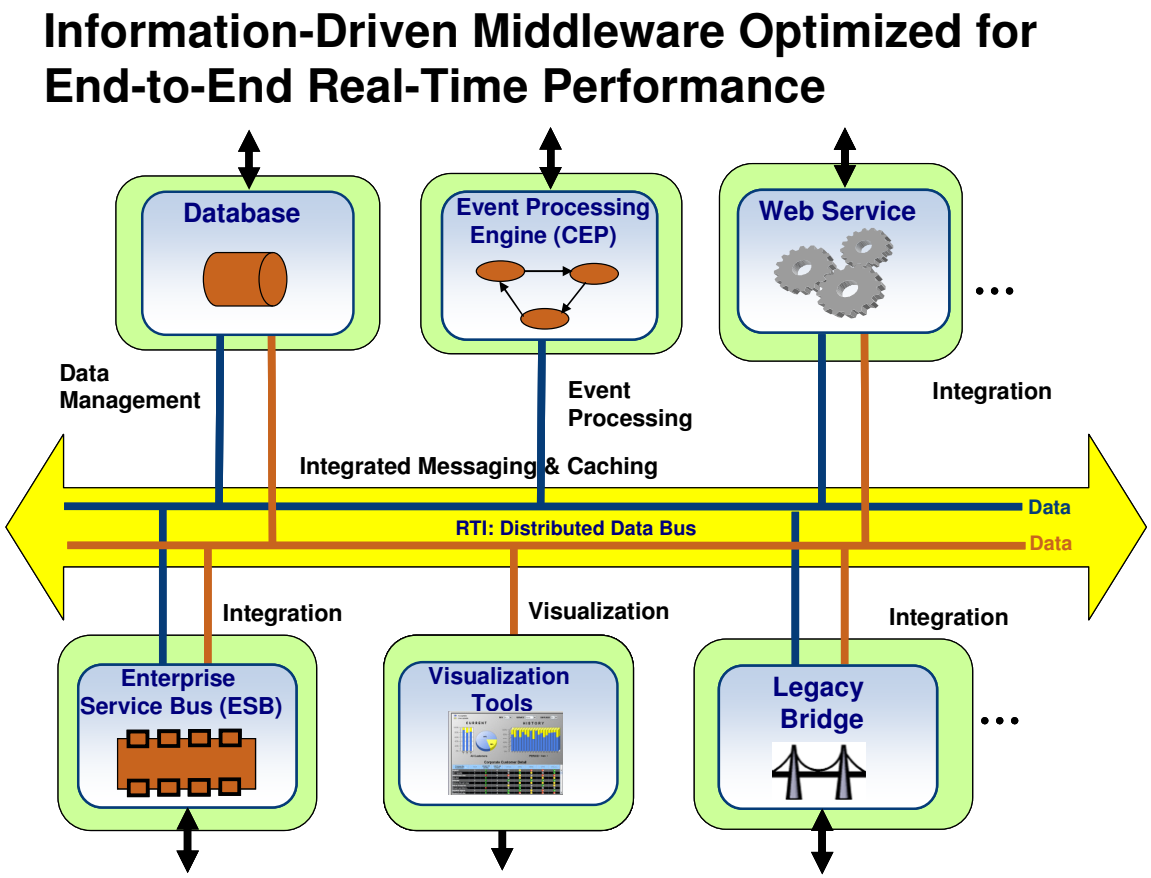

Figure 5 Data-oriented design makes it possible to define automatic mappings for popular application platforms' native data representations to underlying semantic data models. This speeds and optimizes integrations with messaging and caching middleware such as the RTI Data Distribution Service.

Leading real-time middleware vendors have already begun providing a complete ***end-to-end real-time application development platform*** comprised of components that integrate the real-time messaging infrastructure with popular application platform technologies such as (1) event-processing engines for turning raw messages into "intelligent" events in continuously flowing real-time streams; (2) databases to automatically map tables stored in memory or on disk into real-time data sources and/or sinks; and (3) visualization tools to create sophisticated operator dashboards that are continuously updated from real-time feeds.

The availability of such standards-based integrated application components dramatically lowers the risk in software integration and speeds up the development of interoperable, open standards-based ground-station software.

3.3 INTEGRATE AROUND THE SYSTEM INFORMATION MODEL

The data-oriented interfaces of the various components collectively define the *system information* model. It is the basis of integrating new mission capabilities and reconfiguring for a

different mission. The application components rely on the underlying integrated commercial off-the-shelf (COTS) platform to distribute the data per the desired QoS and notify of exceptions.

The next-generation telemetry systems based on a data-oriented architecture reap several benefits, including the ones listed below:

- **Allow for easily modifying and upgrading existing systems**. By standardizing on the data model, it is easier to modify and upgrade systems, since they are no longer tightly coupled and no longer make strong behavioral assumptions about the components they interact with.

- **Provide redundancy and fail-over capabilities**. With point-to-point connections, when a data source or connection is lost, the data consumer loses that information even if it may be available elsewhere on the network. In a data-oriented approach, the data is resilient in that it is abstracted away from the source. If there are multiple sources, data remains available as long as one source or connection remains active.

- **Provide persistence of critical data**. In the dynamic telemetry environment, some data sources may be transient as they become available for short periods of time. The data can be made available to interested consumers even when the original source is not directly available.

- **Offer the ability to distinguish between different types of data**. This allows us to move away from one-pipe-fits-fall communication to multiple individually tuned data flows. The QoS parameters are specified on a per-data-flow basis, allowing for better real-time performance, durability and reliability for each type of information flow.

4 APPLICATIONS

DDS is currently used in hundreds of applications. Some recent examples are outlined on the OMG portal [6]. Historically, military systems were first to aggressively adopt DDS.

Specifically, in the context of telemetry, a data-oriented architecture using DDS has been adopted by a large remote-sensing telescope project, is being considered by a test range and is being evaluated for space applications. Several unmanned systems are now using DDS to support reconfigurable missions that include telemetry [8].

5 CONCLUSIONS

Data-oriented architecture is a framework for creating loosely coupled, real-time, information-driven systems. It emphasizes a common underlying semantic data model as a first-class citizen around which application components are organized. Application components offer data-oriented interfaces, which describe the role (producer/consumer), the data model, the associated metadata

and the associated QoS. Since there is no direct coupling among the application-component interfaces, components can be added and removed in a modular and scalable manner.

The application components rely on a tunable real-time middleware infrastructure that can interpret and manage the underlying data models on their behalf. The tunable middleware infrastructure is responsible for providing optimized, end-to-end, real-time performance between the application components and supporting tradeoffs between competing requirements.

REFERENCES

[1] Rajive Joshi and Gerardo Pardo-Castellote. OMG's Data Distribution Service Standard: An overview for real-time systems. Dr. Dobbs's Portal. Nov 2006. http://www.ddj.com/dept/architect/194900002

[2] Gerardo Pardo-Castellote, Bert Farabaugh, and Rick Warren, An Introduction to DDS and Data-Centric Communications http://www.rti.com/mk/intro_dds.html

[3] Rajive Joshi, Data-Oriented Architecture: Loosely Coupling Systems into "Systems of Systems", RTC Magazine, 2008.
http://www.rtcmagazine.com/home/article.php?id=100926&pg=3

[4] Rajive Joshi. *Data-Oriented Architecture.* Whitepaper. Real-Time Innovations. 2007. http://www.rti.com/mk/data-oriented-architecture.html

[5] Object Management Group. Data Distribution for Real-Time Systems, v1.2 Standard. http://www.omg.org/technology/documents/formal/data_distribution.htm

[6] Object Management Group. DDS Information Day Archive.
http://portals.omg.org/dds/InformationDays/BrusselsJun2007

[7] Stan Schneider, Real-Time Transportation Systems. 7th European Congress on Intelligent Transportation Systems, Geneva, Switzerland, http://www.itsineurope.com/

[8] Rajive Joshi and Stan Schneider. *Architecting an Integrated Reconfigurable Application Platform for UxVs.* AUVSI's Unmanned Systems North America 2008.