How Does Your Real-time Data Look?

By Supreet Oberoi

Real-Time Innovations, Inc.
385 Moffett Park Drive
Sunnyvale, CA 94089
www.rti.com

Introduction

Are all real-time distributed applications supposed to be designed the same way? Is the design for a UAV-based application the same as that of a command-and-control application?

In the case of a UAV-based application, a high volume of data gets created, only some of which is of interest to the base station. To preserve the radio link's bandwidth, only the relevant information is transmitted. The application will use the data for post-mission analysis, so it also has persistence and data-mining needs. In contrast, a real-time command-and-control application needs low-latency and high-reliability, but has little need to persist or cleanse the data in real-time. No, all real-time distributed applications are not designed the same way. While we categorize both these applications as real-time with similar data-transmission characteristics, their architectures and designs vary significantly because the information that they manage and process varies significantly.

This paper characterizes the lifecycle of data in real-time applications—from creation to consumption. The paper covers questions that architects should ask about data management—creation, transmission, validation, enrichment, and consumption; questions that will determine the foundation of their project.

Why is it important to characterize your real-time data?

Historically, architects developing real-time distributed applications have focused more on the technical challenges of *network communication* than on the challenges of *information management.*

For example, when developing an application that collects payload data from a UAV, application architects typically focus on the best way to use a lossy and low-throughput radio link, how to manage an ad-hoc network, how to manage reliability in network transmissions, and other network-specific characteristics.

Until the development of real-time middleware like OMG's Data Distribution Service for real-time systems, these network communication issues posed a significant challenge. By designing a distributed system without understanding the true characteristics of the data, you could end up with an application that wastes valuable resources—from network bandwidth, to CPU, to memory.

For example, an application that monitors sensor alert conditions may waste network resources by not using Complex Event Processing (CEP) to perform local processing of data and by only transmitting alert conditions; a UAV-based application that requires simulation of payload data may incur delays in analyzing data that is not stored correctly in the first place.

To summarize, building a distributed system inherently implies that it is a network communications challenge—in reality, it is also an *information management* challenge.

Question 1: What types of data do you have?

Different types of data have different data transmission, persistence, processing, and mining needs. Again, taking the example of a UAV-based application: the application may exchange or transmit video, sensor (payload) or control data with the ground station. While the network characteristics for sending video data may tolerate loss of data, this will not be true for exchanging control data ("plane, turn right") where the network link needs to be reliable.

The video data will require a high-throughput connection and persistence for later analysis when data is replayed, while the control data may not require persistence. The sensor data may require preprocessing for events and erroneous readings before transmission, but the control data has no preprocessing requirements.

Since the application architect can foresee mining the sensor data, she may require all sensor data to be persisted in a relational database. However, video data will most likely be queried only by timestamps and no other associated dimensions; persisting it in a relational database is not required.

At the very least, the system architect should pose the following questions about each type of data that exists in the system:

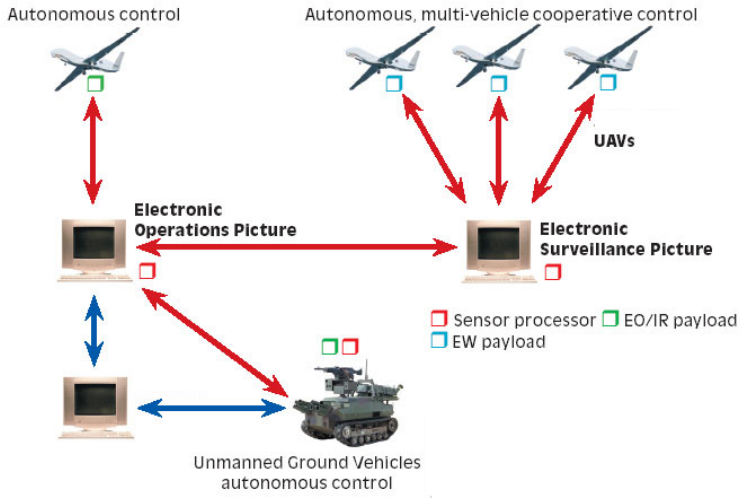

Figure 1: A simplified information diagram of a UAV-based system

What are the network transmission characteristics of your data?

- Rate and volume of data generation
- Data-transmission performance
- Preprocessing and event inference

What are the post-*network transmission characteristics of your data?*

- Persistence
- Replay and simulation
- Data querying and mining
- Integration with enterprise and legacy systems

Question 2: How much and how fast are you publishing data?

Consider the example UAV-based system that is publishing a video feed to a ground station. To design an integrated information management system, the architect should ask these questions:

- How fast is the video data created?
- What is the typical size of a video sample?
- On average, how many publishers are sending data to the ground station?
- For how long does the data need to be available for mining and for query?

Without a proper understanding of answers to these questions, the architect runs the risk of making many mistakes: overwhelming the publishing and/or subscribing node with data, overwhelming the network with data, not using the right information management system like RDBMS, not using data enrichment and summarization techniques like Complex Event Processing.

Question 3: What are your data-transmission performance needs?

The performance characteristics for your data transmission can be different from your network transmission. For example, on a UAV supported by a low-throughput, high-latency radio link, the video feed may have high-throughput performance constraints and the sensor alerts may have low-latency constraints. Similarly, the control data on a command-and-control application running over a gigabit LAN (high throughput, low latency) will have low-latency, but low-throughput, constraints.

Based on the data characteristics, the application architect will configure the middleware differently—for example, to maximize an application's throughput, the developer may use RTI middleware's message batching feature.

While latency and throughput are two key data performance constraints, there are others, such as CPU and memory usage. While embedded RTOS systems continue to obey Moore's law of expanding CPU power, they are traditionally more constrained both by memory and by CPU compared to their enterprise cousins like Solaris, Windows, or Linux platforms. An RTOS system polling a sensor at a high rate with strict demands for reliability will consume high memory resources. (RTOS will keep samples in memory until the subscriber confirms that the data has been received.) Similarly, a system that is publishing large volumes of data will most likely cause a bottleneck with CPU usage.

Question 4: Does your information have any real-time processing needs?

Distributed systems need real-time information processing for many reasons, such as preserving network bandwidth or *reacting* to events in real time. For example, a distributed system that is monitoring for networked-node intrusion will want to do real-time processing of all open network ports to determine if they are authorized. If an unauthorized or suspicious open port is detected, the application will disseminate that event in real time to all other nodes. In other words, the application architecture cannot afford the high latency of sending the information to a centralized node for processing. In addition, the patterns for detecting alerts may change frequently—again, the example of malicious node access comes to mind. Hackers continually apply new strategies to gain unauthorized access, so the algorithms for processing network I/O need to be updated very frequently. In such cases, using Complex Event Processing along with real-time data distribution middleware like RTI Data Distribution Service is useful.

Another reason for doing real-time processing of the data as close to the source as possible is to preserve network bandwidth. A sensor on a satellite may be collecting samples at a very high rate, but perhaps the ground station is only interested when an event of interest happens. In that case, it is prudent to preserve the network bandwidth by doing as much of the information processing as possible on the satellite, and only transmitting events of interest to

the ground station. Again, the architects can benefit by leveraging technologies such as Complex Event Processing to validate and enrich the data before it is prepared for network transmission.

Question 5: What are your application's persistence needs?

Applications engineers and architects who delve in the real-time world of embedded systems and RTOS sometimes overlook the fact that the data needs to be persisted, or that issues pertaining to persistence are equally relevant. The bottom-line goal for most applications—real time or not, distributed or not—is to generate data that can be consumed. In this cycle of data creation and consumption, persistence plays a key part. Application architects can use persistence to achieve many different functional goals, such as fault tolerance, providing historical data for late-joining subscribers, and archival.

Consider the example of a sensor that is generating data with high-reliability constraints. To make sure that no data is lost if there is a crash in the middle of publishing the data on the wire, or to make sure that all data is received if a subscriber crashes in the middle of reading the data from the wire, the application architect will need to employ persistence techniques. In addition, the architect will need to use persistence to enable replay, data mining, or integration (these topics are discussed in the following section).

Depending on the location and the nature of access, different persistence tools will be used. For example, to enable persistence for a relatively low volume of data on a UAV's RTOS board, an embedded database like SQLite may be used. To persist data like video feeds that have relatively simple query interfaces (query by time), even a flat file may suffice. In contrast, the persistence needs for storing sensor payload data from many different UAVs at a ground station will require a more complex system, such as an enterprise-grade RDBMS with sophisticated capabilities to do data mining and managing large volumes of data.

If the application demands the use of a relational database, then the data structures need to be mapped to database tables. The architect will need to resolve throughput-impedance-mismatch issues, since data rates in real-time applications are traditionally much faster than the rates that enterprise-grade databases can manage. The architect may choose to use the RTI middleware's content-filtering and Complex Event Processing capabilities, along with an in-memory database like TimesTen, to address such issues.

Question 6: Does your system have replay or simulation needs?

Every developer with experience in building distributed systems already knows that *debugging distributed applications is tough!* There are many reasons.

First, many bugs in distributed applications are caused by a complex sequence of events between networked nodes that are difficult to reproduce. A particular event may be caused by a machine failure at a particular moment, which is difficult to reproduce. Another reason is that it is very *expensive* to recreate the test case for debugging. For example, there may be an application failure during a field test in deploying a distributing application on a ship, or in running a field test with a set of UAVs. It is just not feasible for an engineer to reserve a ship until she fixes her bug!

One approach that is very useful in debugging distributed applications is to capture all the data—discovery, metadata, and user data—without affecting the performance of the system. Then later, if the developer wants to recreate a particular incident that occurred during a field test, she can replay the captured data in the same order, at the same or different rate, to better understand why the system reacted the way it did.

This capability is different from having persistence for your application. Persistence usually applies to saving the desired set of *user* data. What we are referring to here is the capability to store *all* the relevant data that is required to recreate the state of the distributed application. An application architect may consider using RTI Recorder to address such issues.

Question 7: How does your data integrate?

In a world of mandated interoperability between systems, and a global information grid, application architects need to make sure that *relevant* data is available, or even pushed to an external system.

Application architects need to consider tools and technologies that make integration with back-end

enterprise systems scalable and *flexible* (no system redesign required if a data structure is updated).

Enterprise Service Bus (ESB), with the use of web-services, is increasingly becoming the technology of choice when designing systems for integrations. A Complex Event Processing engine, which contains adapters for integrating different systems like RTI middleware and RDBMS, can also be a sensible choice.

To design for integration with back-end systems, an application architect should address these questions:

- What subset of real-time data do I need to integrate?
- What processing (cleansing, validating, enriching) do I need to perform prior to integration?
- How will I resolve the data-impedance issue between a real-time and non-real time system?
- How do I build a flexible system that loosely couples the real-time system with the back-end system?
- Do I need to use ESB or a CEP?

Summary

The goal of building a distributed, real-time application is the same as the goal of building an enterprise OLTP, CRM, ERP, or any other application—to manage *information* to give your enterprise a competitive edge. To achieve this goal, system and application architects need to focus on information management in addition to network communication and management.

Some of the best answers are provided by asking the right questions. This paper details questions architects should consider addressing as part of their system design.

About RTI

Real-Time Innovations (RTI) provides high-performance infrastructure solutions for the development, deployment and integration of real time, data-driven applications. RTI's messaging, caching, Complex Event Processing (CEP) and visualization capabilities deliver dramatic improvements in latency, throughput and scalability while slashing cost of ownership. The company's software and design expertise have been leveraged in a broad range of industries including defense, intelligence, simulation, industrial control, transportation, finance, medical and communications. Founded in 1991, RTI is privately held and headquartered in Santa Clara, CA. For more information, please visit www.rti.com.

v0609b