REAL-TIME TENA-ENABLED DATA GATEWAY

Dr. Joachim Achtzehnter and Preston Hauck
NetAcquire Corporation
www.netacquire.com

ABSTRACT

This paper describes the TENA architecture, which has been proposed by the Foundation Initiative 2010 (FI 2010) project as the basis for future US Test Range software systems. The benefits of this new architecture are explained by comparing the future TENA-enabled range infrastructure with the current situation of largely non-interoperable range resources.

Legacy equipment and newly acquired off-the-shelf equipment that does not directly support TENA can be integrated into a TENA environment using TENA Gateways. This paper focuses on issues related to the construction of such gateways, including the important issue of real-time requirements when dealing with real-world data acquisition instruments. The benefits of leveraging commercial off-the-shelf (COTS) Data Acquisition Systems that are based on true real-time operating systems are discussed in the context of TENA Gateway construction.

KEYWORDS

TENA, Gateway, Data Acquisition, Real Time, Distributed Systems

THE PROBLEM

Historically, US Test Ranges have evolved autonomously. This has resulted in considerable duplication of effort, inconsistent processes and procedures, and many stovepipe systems that don't interoperate easily. These issues are becoming increasingly problematic for the range community as more and more tests now involve multiple test ranges. This trend highlights the need for standardization, both in terms of technological interfaces as well as in terms of procedures.

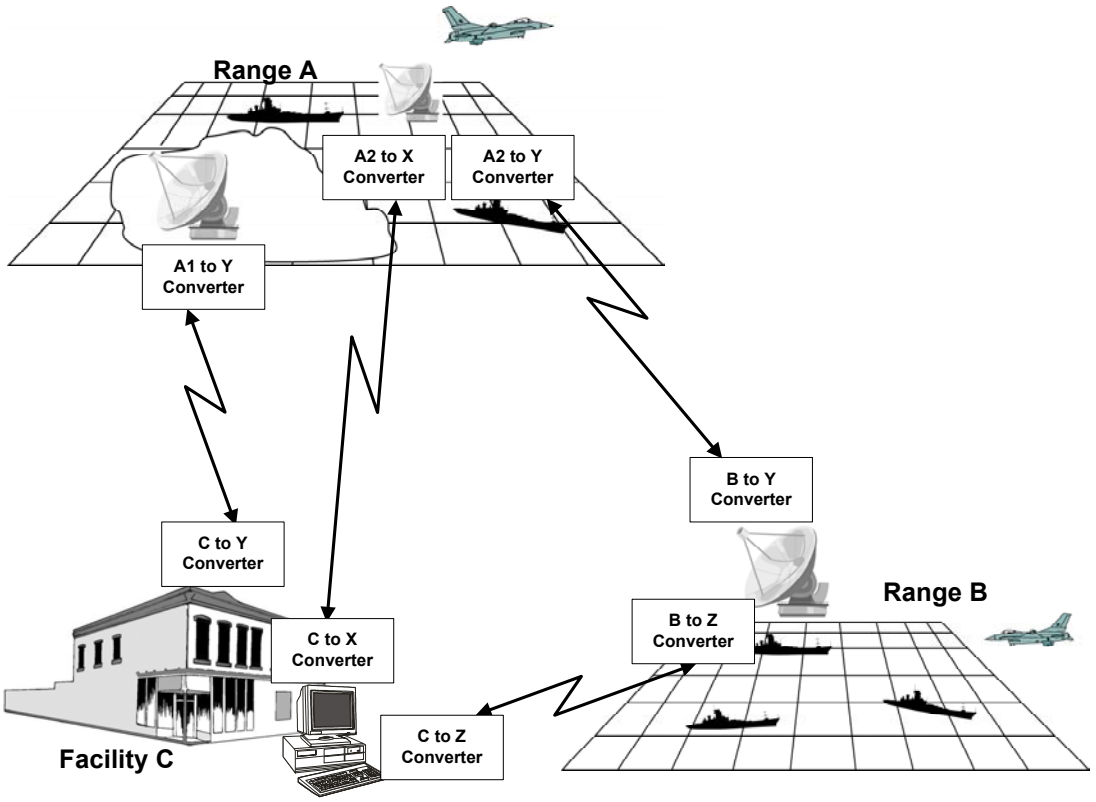

Figure 1 - Current Approach

Figure 1 illustrates the current situation where tests involving multiple ranges almost invariably require custom solutions to get resources at different ranges to interoperate. The same issues arise to a lesser extent also within a single range, when existing individual resources, including new as well as legacy equipment, are utilized together in new combinations.

TENA TO THE RESCUE

The Foundation Initiative 2010 project attempts to address these problems with a new common architecture for range resources to provide [1]:

- Cost-Effective Interoperability between test resources
- Cost-Effective Development and Maintenance of Instrumentation Systems

This new common architecture was given the name Test & Evaluation Network Architecture [1], or TENA for short. In later documents the TENA acronym is also translated as Test and Training ENabling Architecture [2], probably reflecting the fact that the training community was now considered as another important target audience for the TENA architecture.

Special emphasis was placed on the requirement to support modular designs with the capability to allow range engineers to add “plug-in” features specific to their applications [2]. Furthermore, the incorporation of existing range resources, including legacy radar systems, was to be maximized.

The design of the TENA architecture [3] is based on a new concept called Stateful Distributed Object, or SDO[1]. The SDO concept fuses two software design paradigms into one single framework. The two paradigms in question are distributed objects and the publish/subscribe paradigm. Separately, they have been widely adopted for distributed applications, but TENA is probably the first major architecture that has been able to marry these two ideas into an integrated whole.

Object technology has been a dominant force in the software world for many years, especially in the enterprise and consumer software markets, perhaps somewhat less so in the real-time and embedded systems markets. The main strength of object technology is its support for abstraction that allows designs that are expressed in terms of concepts that are meaningful to the application's users. By reducing the conceptual gap between implementation-level and user-level entities, the resulting designs are easier to adapt as user-level requirements change. Communication between system designers and implementers on the one hand, and users and procurement staff on the other hand is also facilitated. Several distributed object frameworks have found widespread application, such as CORBA and DCOM. As these technologies matured they have also become more suitable for the more demanding applications with real-time requirements. An example of this development was the adoption of a Real-Time CORBA standard, and the availability of several compliant products.

The publish/subscribe paradigm also has found widespread use in the real-time and embedded systems market with several mature middleware products [4, 5] that are targeted especially at these markets. The two main benefits of this paradigm are the logical decoupling of data sources from data sinks, and the guaranteed low-latency access to the published data values. The low-latency aspect explains the popularity of publish/subscribe for distributed applications with real-time requirements.

By combining these two well-known and well-established paradigms for designing distributed applications, the Stateful Distributed Object framework can offer the advantages of both paradigms without burdening application designers with the distinct disadvantages associated with each paradigm when either is used in isolation.

A Stateful Distributed Object exposes a remote method interface analogous to a CORBA or DCOM interface. The distributed nature of an SDO is reflected in the fact that at runtime there is one SDO implementation instance and zero or more remote proxy SDO instances through which client applications can invoke the interface's operations. The proxies forward or delegate these invocations over a network to the implementation instance.

An important addition to an SDO is the notion of a publication state. The publication state is a collection of data items. The latest known values of these data items are cached by the SDO proxy, hence allowing low-latency access without incurring the overhead of a remote invocation for every access. The data that comprises the SDO's publication state is disseminated by the SDO's implementation instance on its own initiative and without explicit knowledge about the possible existence of proxies. This is managed transparently by the TENA middleware layer on

[1] The TENA architecture also introduces messages and streams as basic concepts. This paper focuses exclusively on the more important SDO concept.

behalf of the SDO implementation. Figure 2 illustrates the interaction between an SDO Proxy and its corresponding SDO Servant.

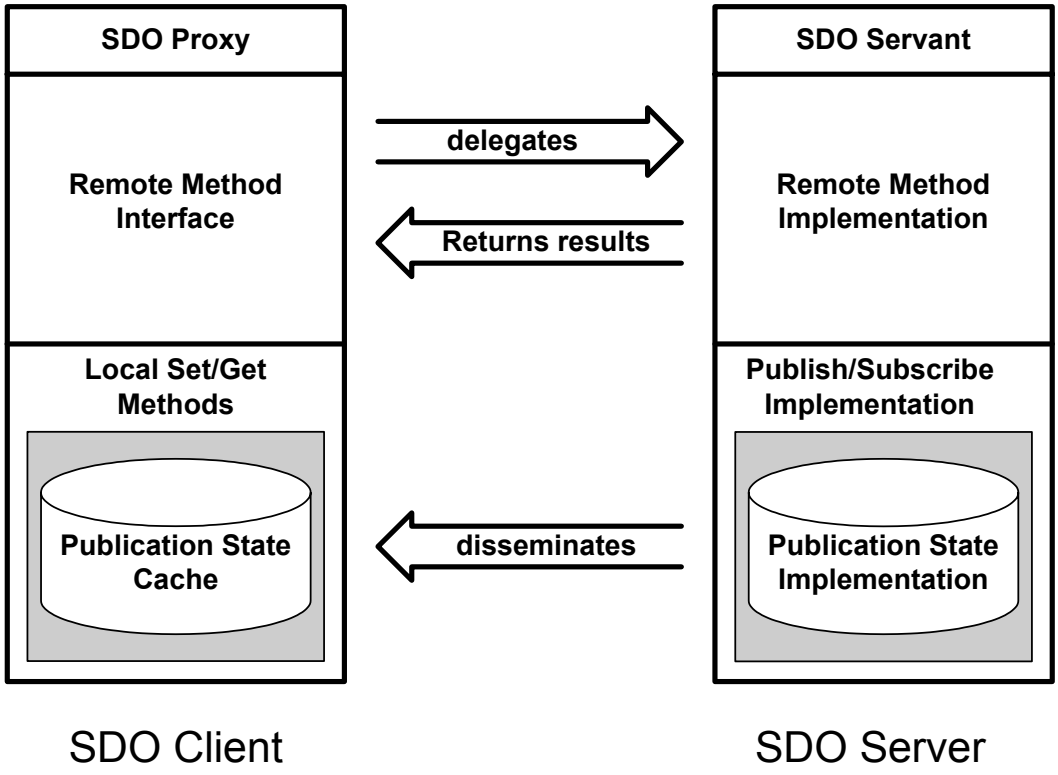

Figure 2 - Stateful Distributed Object

From the point of view of real-time applications specifically, the notoriously long delays associated with remote method invocations and their unpredictability is avoided when accessing a remote SDO's publication state. This is achieved without sacrificing the high-level modeling capability of object-oriented designs. Unlike typical low-level designs that result from a purely data-centric approach so often seen when publish/subscribe middleware is used by itself, designs in terms of Stateful Distributed Objects can benefit from the advantages of object technology with its abstraction capability without giving up low-latency access to the publication state of remote objects.

The interface of a particular type of SDO, including its publication state, is defined using a platform-independent and programming language neutral specification language called TDL, the TENA Definition Language. The interface definition expressed in TDL serves as the contract between the SDO provider and its users. The plan is that TENA will provide several types of SDOs that have general applicability in the test range community. Additionally, the extensible design makes it possible to add application-specific SDOs to address requirements not covered by the standardized SDO types.

The process of standardization is expected to evolve over time, with many SDO designs first created as application-specific or location-specific custom extensions. A subset of these may prove generally useful and can then become part of a standard object model. An object model is a coherent set of collaborating SDOs that address a particular application domain.

The FI 2010 project has created a reference implementation of the TENA architecture. This reference implementation is internally based on CORBA, a widely used distributed object middleware standard. The remote method interface of the SDO abstraction is mapped more or less directly to CORBA interfaces. The publication state concept, the other important ingredient of the SDO abstraction, is internally implemented in terms of CORBA remote invocations using the CORBA Notification Service.

TENA-ENABLING LEGACY EQUIPMENT

One of the original requirements for the TENA architecture was the ability to incorporate legacy equipment into the new infrastructure. This can be achieved by introducing so-called TENA Gateways. Such a gateway communicates with the legacy equipment using that equipment's native API and makes the same functionality available in a TENA-compliant fashion by implementing one or more Stateful Distributed Objects.

Such a gateway can use either a special-purpose system expressly designed for a particular piece of legacy equipment, or can use a general purpose, TENA-enabled data acquisition system. Either way, depending on the nature of the legacy equipment that is to be represented by the gateway, the TENA Gateway often must satisfy strict response time requirements. In other words, such instrumentation typically requires servicing within hard real-time deadlines to avoid data loss or worse consequences.

The original reference implementation of TENA has been made available by the FI 2010 project for several platforms including MS Windows and GNU/Linux, but none of these platforms are suitable for hard real-time applications. Real-time systems must respond in a timely (predictable) way to external events. System correctness depends not only on the logical result of computations but also on the delivery time (latency) of the results. Note that real-time is not the same thing as "real-fast". In particular, Windows and Linux systems can be fast but also have poor real-time behavior (occasional worst-case response time can be one hundred times worse than typical response time). Various attempts to extend Windows and Linux to support real-time requirements have met with only limited success.

Real-time general purpose Data Acquisition Systems such as those manufactured by NetAcquire Corporation are designed to be configured without programming for a variety of specific data acquisition and control tasks. These devices run a true real-time operating system and as such they are a convenient base on which to build TENA Gateways.

Given that the TENA reference implementation was based on standard components such as the ACE/TAO CORBA middleware product and ISO Standard C++, there is no reason in principle why TENA cannot be ported to a real-time operating system.

As a proof of concept [6], NetAcquire has ported the TENA reference implementation (Version 3) to the NetAcquire real-time platform. This was accomplished on a surprisingly short time

scale. The ease of porting was a consequence of several factors, an important one being the fact that TENA is well designed and reflects a consistent architectural vision. Also very helpful was the fact that the NetAcquire platform already included advanced support for CORBA as well as the ACE/TAO middleware product specifically. Thus, the NetAcquire TENA-enabled system provides not only a real-time operating system, but also other critical software infrastructure components needed to host a distributed data acquisition application (I/O device support, embedded Web server, dynamic network and data buffer management, name server, etc.). During the TENA port and subsequent deployments, the NetAcquire platform did in fact prove itself to be an effective host for real-time TENA.

RAPID TENA SDO DEVELOPMENT

In addition to porting the TENA middleware implementation, several SDO implementations were created to demonstrate the TENA functionality. Two standard Stateful Distributed Object types were implemented: a Time Space Position Information (TSPI) object and a Radar object (both object models have been used by the TENA community for other TENA proof of concept demonstrations).

The TSPI object reported the GPS time and position information obtained from an attached GPS antenna.

The Radar object model leverages the NetAcquire system's ability to natively communicate using synchronous serial protocols to legacy radar equipment. This Radar SDO was implemented as a NetAcquire Data Flow extension [7] and thereby served to illustrate an important advantage of utilizing a powerful, general-purpose data acquisition system to create TENA Gateways. As a Data Flow component, the Radar SDO does not need to be hard-coded to support a specific model of radar equipment. Instead, the NetAcquire Data Flow Designer (a graphical data flow design tool) can be used to interactively describe the input and output needs of a particular legacy radar system. The most important aspect in which different radar systems can differ is the serial frame format. This frame format can be defined easily in the Data Flow framework using a graphical frame map editor.

Figure 3 shows an actual data flow diagram with the embedded Radar SDO object. The legacy radar serial stream enters the data flow via the Pull Reader object, the Decommutator object extracts individual radar data values from their respective fields in the serial data frame, a Computation Set performs any required transformations, for example engineering unit or coordinate transformations, and the results are finally fed to the TENA Radar SDO implementation.

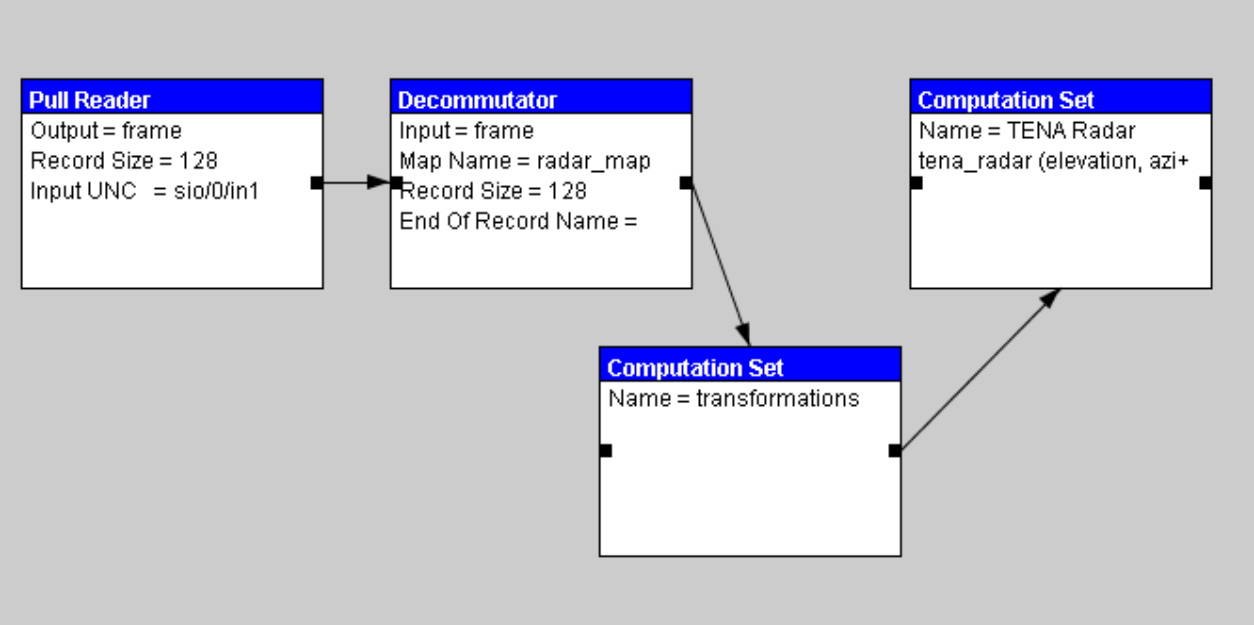

Figure 3 - TENA Radar object within a NetAcquire Data Flow

TENA WITHOUT PROGRAMMING

As a demonstration of the versatility of the TENA architecture and specifically the benefits of a TENA-enabled general-purpose, real-time data acquisition system, a custom TENA object model was created that represents generic measurements in an object-oriented way (see Figure 4). The Measurand SDO, just like the Radar SDO, was implemented as a NetAcquire Data Flow extension with the name "TENA Publisher". This TENA Publisher is a drop-in replacement for the standard Publisher component provided with NetAcquire products. The latter publishes measurands using the NetAcquire Publish/Subscribe API, which is a low-latency, push-driven mechanism for publishing individual measurements. The NetAcquire TENA Publisher performs the same function but uses the TENA Publication State concept to disseminate measurement updates to all interested TENA clients.

With the Measurand object model and the corresponding NetAcquire TENA Publisher SDO implementation, a real-time special-purpose, TENA-enabled Data Acquisition system can be configured within minutes without programming. Measurement data can be acquired using any combination of many supported NetAcquire I/O devices, such as analog and digital I/O devices and telemetry signals of different formats including IRIG PCM and CCSDS multiplexed virtual and packet channels.

A typical Data Flow configuration using the TENA Publisher component is shown in Figure 5. Here telemetry frames enter the Data Flow on the left via a PullReader, are decommutated into individual measurands by a Decommutator component, which are then published by the TENA Publisher component. The name of an input data frame map is a configuration parameter to both the Decommutator and the TENA Publisher. The Decommutator uses the information in the map to determine how individual measurands should be extracted from the input frame and the TENA Publisher obtains the measurand names, types, and units from the telemetry map.

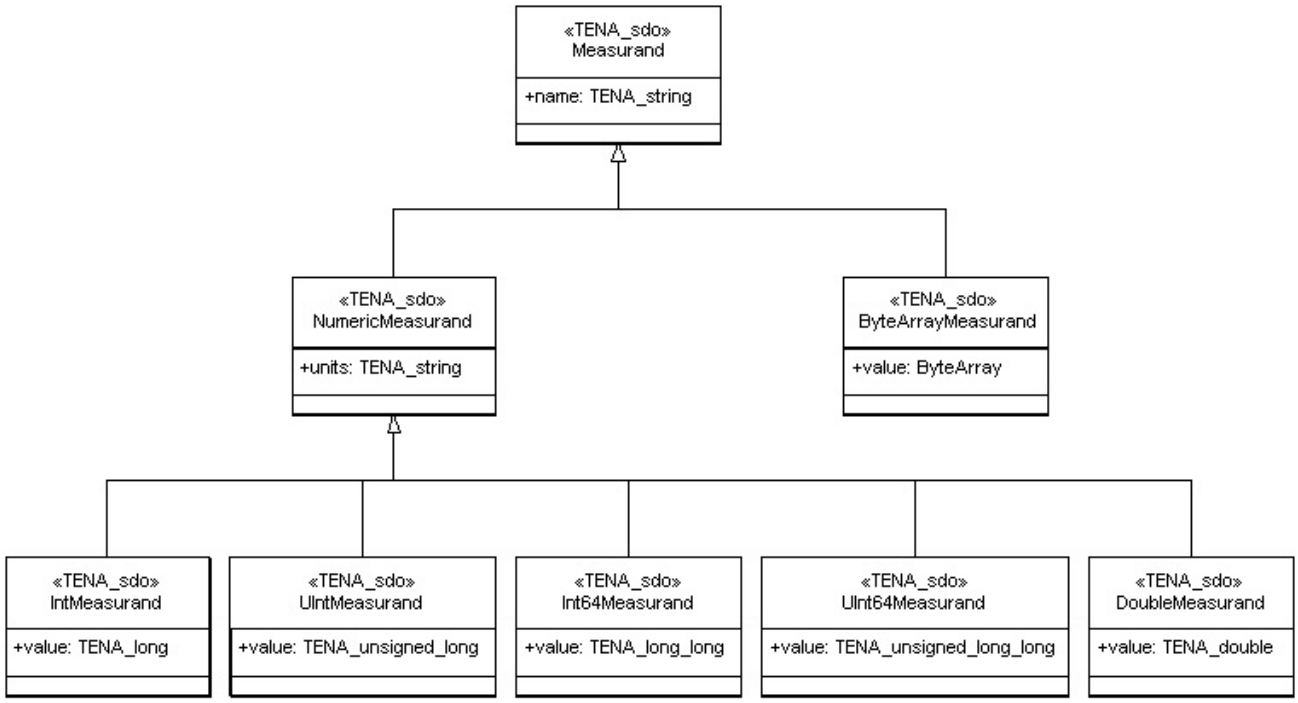

Figure 4 - The Measurand Object Model

The same Data Flow configuration will publish analog or digital data acquisition inputs. The only difference is that the PullReader would be configured to read from the appropriate data acquisition device instead of from a serial I/O device, and the format of the input frames as described by the "telemetry" map is typically simpler in this case.

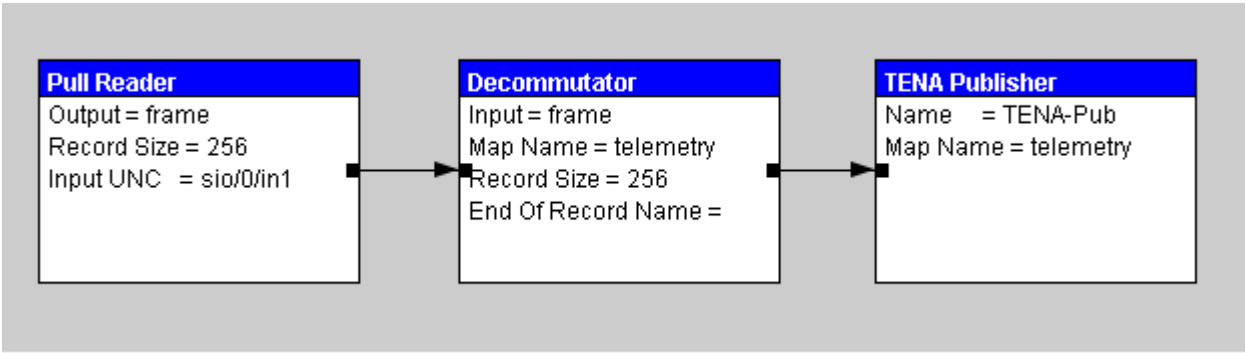

Figure 5 - TENA Publisher in a Data Flow

REAL-TIME CONSIDERATIONS

TENA gateways are often required to run in real-time. That is, the gateway must guarantee that both a fast and a deterministic response time as data passes through the gateway. As described earlier and in other references [8], general purpose desktop operating system like Microsoft Windows or Linux cannot meet strict real-time constraints.

In order to address the requirements of real-time, a real-time time operating system (RTOS) is normally employed. A true real-time operating system is designed from the very start with the primary goals of having the highest possible performance in areas such as:

- Very fast Interrupt Service Routine (ISR) response time
- Pre-emptive, priority-based multi-threading at the kernel layer
- Fast task switching time
- Priority-aware mutex, semaphore, and other inter-process communication/scheduling mechanisms
- Use of non-virtual memory for predictable and timely memory management
- Small thread time slices

As an example, Windows typically services each process at about 200 Hz (every 5 milliseconds). In contrast, an RTOS such as is used on a NetAcquire system, can time slice at up to 50,000 Hz (every 20 ***micro***seconds), a factor of 250 times improvement.

In addition to the RTOS itself, other software components must be specially architected to preserve real-time performance.

One such case is the device driver layer. Software drivers need to be written to minimize the adverse effects of excessive data buffering. Network hardware device drivers are a typical example. Operating systems support network peripherals like Ethernet with a number of software-implemented layered protocols called a network "stack". The network stack that comes with a desktop operating system is not real-time responsive, in that it is not priority based and typically does large amounts of buffering. RTOS vendors are well aware of this problem and use network stacks that are carefully designed to not introduce any delay, despite the existence of other active tasks, network retransmits, and variable latency at the computer interacting on the other end of the network.

Another example is at the applications level, where it is important to take advantage of fine-grained thread priorities and priority-based synchronization to minimize latency. A TENA gateway can consist of close to 30 cooperating threads—without priority-driven thread scheduling of a real-time application environment, data latency is significantly impacted.

GENERAL PURPOSE REAL-TIME TENA GATEWAY PLATFORM

Encouraged by the positive reception that these proof of concept implementations have received in the TENA community, NetAcquire Corporation is developing a full product release of a NetAcquire-hosted TENA middleware library based on the latest version of the TENA reference implementation (Version 4). This will include a TENA Programmer's Toolkit comprised of TENA middleware header files, source files, and libraries that are specially configured for and ported to the NetAcquire real-time platform. In addition to the generic middleware support for local SDO implementations and for invoking remote SDO instances, there will be support for session and execution management. Local SDO implementations can utilize the full range of

SDO features including attributes and remote methods. User-supplied object models can be integrated with the help of a supplied TDL compiler.

A version of the Measurand object model described previously will also be included to allow the creation of custom Data Acquisition systems without any programming required. Feedback from customers who tested the Measurand object model in its proof of concept incarnation was especially enthusiastic, and consequently it is possible that this object model or a modified version of it will become a standard TENA object model in the future.

CONCLUSION

This paper has shown several examples of leveraging the advantages of a true real-time Data Acquisition platform for the construction of TENA Gateways with minimal effort. Compared to the alternative of creating a "one-off" special-purpose TENA Gateway for every different piece of legacy equipment on general-purpose, desk-top operating systems, the approach proposed in this paper not only has the advantage of saving development time and costs, but at the same time can better ensure that the real-world equipment represented by the gateway will be serviced correctly in a timely manner, especially when hard real-time deadlines must be maintained.

REFERENCES

1. Joint Overarching Requirements Document for Foundation Initiative 2010 (FI 2010), 31 October 1998.

2. Test Capabilities Requirements Document for Foundation Initiative 2010 (FI 2010), 31 October 2000.

3. TENA Architecture Reference Document, 4 November 2002.

4. NetAcquire Publish/Subscribe (http://www.netacquire.com/ps.htm)

5. RTI NDDS (http://www.rti.com/products/ndds/index.html),

6. NetAcquire Technical Note NA-TN-052, TENA Support on the NetAcquire Platform, 19 February 2004.

7. NetAcquire Data Flow, http://www.netacquire.com/ data_flow.htm.

8. Improving Real-Time Latency Performance on COTS Architectures, ITC 2003, http://www.netacquire.com/whitepapers.htm.