

Smart Grid Communications: QoS Stovepipes or QoS Interoperability?

David E. Bakken, Washington State University, Pullman WA USA bakken@eecs.wsu.edu

Richard E. Schantz, BBN Technologies, Cambridge MA USA schantz@bbn.com

Richard D. Tucker, Tucker Engineering Associates, Locust NC USA, richardaet@aol.com

Keywords: interoperability, middleware, quality of service, QoS stovepipes, critical infrastructures.

Abstract Interoperability is a key requirement for data communications in the “smart grid”. It has been articulated at great length by the GridWise Architecture Council (GWAC). However, the interoperability issues identified here to date include only interoperability of the data exchange. In this paper, we first argue that middleware is a key enabling technology for helping meet interoperability requirements and avoid stovepipe systems in the smart grid. We then argue that the smart grid’s data communications must support interoperability of Quality of Service (QoS) and security mechanisms across an entire power grid; this will necessarily involve traversing multiple organizations’ IT infrastructures that may have different network-level mechanisms for providing QoS and security. We introduce the concept of QoS stovepipes to help illustrate how such QoS and security interoperability may occur and its consequences. We then argue that the application programmer interface for such QoS and security requirements must be kept as high-level as possible to avoid QoS stovepipes. Finally, we argue that middleware-level mechanisms are a much better way to provide this end-to-end QoS and security, compared to the usual technique in the power grid of using (and getting locked into) network-level mechanisms (which the middleware is built on top of).

1. INTRODUCTION

Interoperability both within and across utilities is a major concern as the communications systems for the “smart grid” are being envisioned and planned [1, 2, 3, 4, 5, 6]. Two major categories of interoperability are network interoperability and syntactic interoperability [3]. *Network interoperability* involves “exchange of messages between systems across a variety of networks”. *Syntactic interoperability* involves “understanding of data structure in messages exchanged between systems”, typically via network messages.

There are a number of cross-cutting interoperability issues that span multiple interoperability categories [3]. Two key

ones which applications require are “*Security & Privacy*” and “*Quality of Service*”.

Additionally, we believe that any implementations supporting interoperability must also support the following principles articulated in [2]:

Principle I09: An interoperability framework must be practical and achievable:

- Meets performance requirements
- Is reliable
- Is scalable
- Has sufficient breadth to meet the range of business needs

Principle I10: An interoperability strategy must accommodate the coexistence of and evolution through several generations of IT standards and technologies that will reside at any point in time on the Grid.

Middleware (defined further below) is a software technology for organizing and programming distributed applications, that is, those applications with parts of a program or service separated by a network [7]. It has been considered “best practices” in other industries for many years (see, for example, [8, 9]) but has barely been deployed in electric power grids to date, at the least for power applications requiring wide-area situational awareness or to

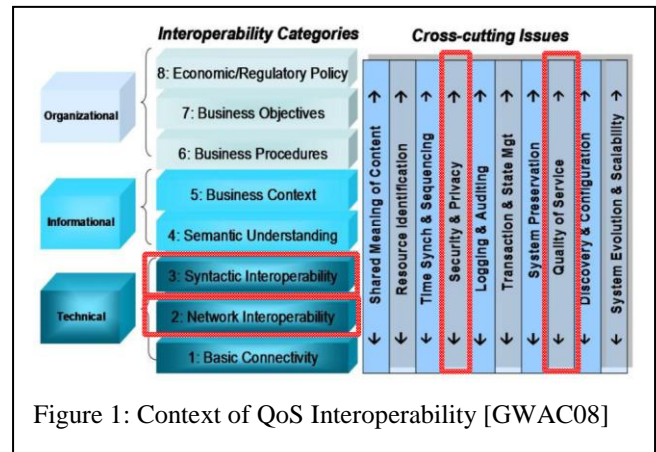


Figure 1: Context of QoS Interoperability [GWAC08]

otherwise augment SCADA.

In this paper, we articulate some issues that must be addressed in support of cross-cutting security and privacy, as well as QoS aspects, in the GWAC network and syntactic interoperability categories. These are highlighted in Figure 1.

The major points of this paper are as follows:

- In support of **Principle I09**, which essentially states that any solutions need to be effective from a variety of operational perspectives, it is essential that syntactic interoperability is incorporated across the board to its individual elements. We believe that having a comprehensive middleware architectural framework to deliver these services is the most effective way to ensure this in a comprehensive way, instead of a large collection of individual but narrow approaches, mechanisms, and evaluations.
- In order to support interoperability across organizations and in support of the "future proofing" articulated in **Principle I10**, it is essential that APIs for Quality of Service (including security) should be expressed at a middleware layer, which maps down onto the lower-level mechanisms for providing a given property, in order to extend life cycle management across the evolution of these mechanisms.
- In order to support multiple non-functional/QoS properties (delay, rate, confidentiality, criticality/availability, ...), it is essential that APIs be expressed in middleware so that they can be integrated and co-managed.
- We extend the definition of a Stovepipe system to include non-functional properties such as QoS and security. We call this a QoS Stovepipe System, something that the "smart grid" must avoid.

Finally, we note that we use the term 'QoS' in the title and the summary points above to be inclusive of a wide range of non-functional/behavioral requirements for smart grid communications, i.e., including cyber-security issues, rate, synchronization, etc [10]. This is the typical view of many (though not all) applied computer scientists, especially those working in the middleware (vs. the network communication) spaces (see for example [11]). We note, however, that GWAC and others often take a narrower view on what QoS is, so throughout the paper we refer to these properties either as "non-functional properties" or "QoS and security/privacy" or something similar.

The remainder of this paper is organized as follows. Section 2 discusses functional interoperability and its opposite

effect, stovepiped systems. Section 3 analyzes how middleware can enhance functional interoperability, i.e. traditional APIs and contracts. Section 4 discusses non-functional (QoS and security/privacy) issues and interoperability. Section 5 analyzes how middleware can enhance QoS interoperability to avoid creating systems that are stovepipes in terms of end-to-end QoS and security/privacy. Section 6 concludes the paper.

2. FUNCTIONAL INTEROPERABILITY AND STOVEPIPES

We now provide a definition of a *stovepipe system*:

Stovepipe System: a legacy system that is an assemblage of inter-related elements that are so tightly bound together that the individual elements cannot be differentiated, upgraded or refactored. The stovepipe system must be maintained until it can be entirely replaced by a new system [12, 13].

Stovepipe systems are commonplace in many long-lived systems, particularly the military, and unfortunately many examples abound in today's electric power grid. However, they are very expensive to maintain, and the opportunity cost of their inability to be upgraded or refactored is staggering. It is thus essential that as we move forward the smart grid minimize the likelihood of creating stovepipe systems.

Interoperability is of course crucial across multiple organizations, vendors, standards, locations, and time scales. A smart grid clearly requires this [1, 3]. Fortunately, technologies and technical approaches developed in computer networking distributed computing, and software engineering in the last two decades enable the smart grid to be built in ways that can avoid both stovepipe systems *and* vendor lock-in.

Network interoperability (Category 2 in [3]) includes transferring data across different networks, inter-domain naming issues, etc. For example, IP can operate above local area networks (LANs) running different technologies such as Ethernet and token ring. It also encompasses OSI Layers 3 and above¹: network (3), transport (4), session (5), and even sometimes aspects that were formerly thought of as

¹ We note that [3] does not list OSI Layer 6: Presentation. We believe that this evolving document needs to incorporate considerations in this dimension as well. For example, this is where interoperability between different CPU types (big endian vs. little endian) is handled. We do believe, however, that this would be better handled in a Syntactic Interoperability layer (Category 3 of [3]), ideally via common middleware.

part of the application layer (7)². Another example of Network interoperability is the 2008 North American Distribution Metering Standard ANSI C12.22. It is designed to operate with different legacy Electric Utility AMI technologies, including IP, by providing services (including name service) above the transport layer allowing it to ride on any and all networks. [14] ANSI C12.22 represents the OSI session layer (5, 6) and interfaces with ANSI C12.19 representing OSI application layer (7).

Before we can offer further analysis, we must distinguish between two kinds of interoperability: functional interoperability and non-functional interoperability.

Functional interoperability involves the traditional interoperability of the application or "business logic". Functional interoperability of course requires some kind of agreement on the interface: an API or *contract* [3 Sec 2.1]. In the next section we describe how middleware can greatly aid in functional interoperability.

Non-functional interoperability involves interoperability across behavioral issues such as delay and security. In Section 4 we describe this further, then in Section 5 we show how QoS-enabled middleware can aid in providing end-to-end QoS that spans multiple organizations, underlying lower-level QoS mechanisms, etc.

3. MIDDLEWARE AND FUNCTIONAL INTEROPERABILITY

3.1. Definition and Benefits of Middleware

We now provide a definition of *middleware* [7, 15]:

Middleware: a layer or layers of software and services above the operating system but below the application program providing a common programming abstraction and system model across a distributed system.

An example of middleware (of a client-server variety) is given in Figure 2. The middleware API offers a given programming abstraction (distributed objects, distributed tuples, etc) that the middleware implements on top of the given operating system's APIs. This ability for the programmer's API to be shielded from that of a lower-level operating system mechanism (or, as we will see in Section 6, being separated from lower-level non-functional APIs for such QoS/security properties as delay, throughput, and

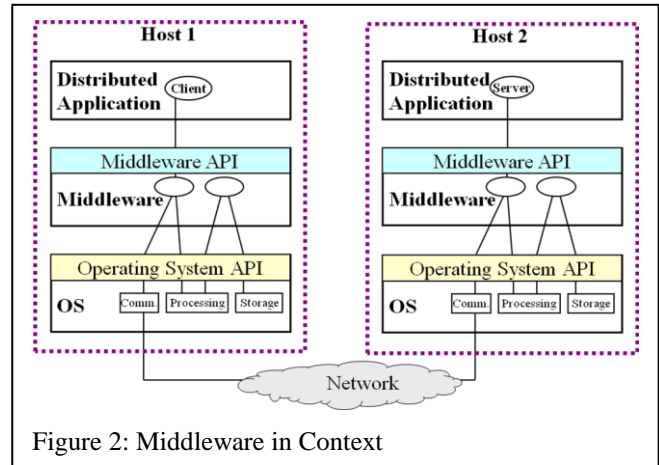


Figure 2: Middleware in Context

confidentiality) is a key advantage of a middleware approach.

Middleware exists in part to help manage the complexity and heterogeneity inherent in distributed systems. Indeed, important aspects of it were developed in large part under the umbrella of the US military, due to its extensive operational needs, e.g. the Cronus project [16]. Middleware provides higher-level and network-centric building blocks (abstractions such as distributed objects, distributed tuples, remotely updated variables, etc) than an operating system provides, and more suitable to distributed computing. Such higher-level building blocks can help make code much more portable, with fewer errors (no need to handle low-level communications issues), and are much easier to change later. They also make programmers more productive because they are much closer to the application's layer of abstraction (servers, publishers, variables, hierarchically-named objects, etc) than the network layer (buffers, DNS names, etc).

Middleware thus also helps insulate the developer from different kinds of heterogeneities that are inherent in a distributed computing system (and provides interoperability across them):

- Network technology (OSI Layers 2 and 3)
- CPU architecture (big/little endian, word size, ...); including OSI Layer 6 (Presentation)
- Operating system (or family thereof; exception: Microsoft Windows -- *de facto* albeit not *de jure*).
- Programming language
- Vendor implementation (some middleware standards, notably from the Object Management Group, have been supported by multiple vendors for many years).

² We note that, in our experience, layers in a distributed computing system often don't look anything like or behave like classically described OSI layers 5 and above; we list them here because they are explicitly used as guidance in [3].

3.2. Standardization of Middleware

Middleware is typically standardized through a combination of at least three levels, to achieve different forms of interoperability:

- API/contract, typically through a language-independent interface definition language (IDL). An IDL compiler then translates the IDL into different supported programming language interfaces, which the programmers program to in the programming language for their particular application.
- Wire protocol: how a method name, server identification, parameter list, etc are marshaled into a network packet
- Message protocol: what kinds of message are sent at a given middleware layer. For example, the CORBA standard specifies 7 different kinds of message; example: a CORBA::REQUEST message goes from the client to the server, which responds back with a CORBA::REPLY message.

Middleware is standardized at the interface levels above, but most often *not* at the implementation level underlying that interface (for most of the implementation; however, key aspects of interoperability e.g. message and wire protocols, also accompany the standardization to achieve more specific interoperability co-objectives.) This allows different vendors to optimize their implementations in different ways and in general to be able to "build a better mousetrap". This in turn makes it feasible for long-running software systems to be able to switch vendors mid-life if needed (e.g., by a vendor bankruptcy or availability of a better implementation). This switch is not without cost, of course, but is a (sometimes small) fraction of the cost of re-implementing the system from scratch.

3.3. Multi-Layered Middleware

Network researchers have of course developed multiple layers of network protocols, where each layer builds on the one below and offers a higher-level of abstraction or service. Similarly, middleware researchers have developed multiple layers of middleware that build on the layer below it [17, 18, 19]. As the layers move higher, not only are the abstractions and services offered to the programmer at a higher level, but they may also become more domain-specific (but still reusable across applications).

3.4. Middleware in Other Industries

The US military has long had complex distributed software spread out over wide geographic areas in hostile and changing IT conditions. It was thus a leader in pushing the

development of middleware; for years, middleware has been required by a number of agencies for their distributed application programs, including the following:

- [US DOD DISA DISR](#) (current DISR baseline version is 09-2; requires DoD PKI Cert to access)
- [US Navy NESI](#) (see NESI-X Part 5 Developer Guidance Mid Tier)
- [US Navy Open Architecture Computing Environment](#) (OACE)
- [US Navy FORCEnet Reference Architecture](#) (FORCEnet Architecture and Standards: Volume II, Technical View)
- US Navy PEO IWS Objective Architecture Software Design Document (Draft)

For more examples of the widespread use of middleware in other industries such as aviation, transportation, and aerospace, see [8].

3.5. Middleware and the Smart Grid

Note that middleware typically overlays and enhances OSI Layers above the transport (4) layer⁴. The alternative to handling these layers in middleware is to hand-code these layers in the application program. However, this is very time-consuming and error prone; the best practices are very hard to re-create [9].

We thus believe that in the layers described in [3], the Network Interoperability layer should address OSI Layers 4 (transport) and below, not including "Application Protocols" as [3 suggests], and arguably not Layer 5 (session), which middleware typically handles. Additionally, we believe that the Syntactic Interoperability layer should encompass issues associated with OSI Layers 5 and 6. ANSI C12.19 was a first effort in Distribution Metering toward solving some of the issues that Middleware has solved in other industries, such as CPU architecture (big/little endian, word size, data types, syntax, syntax organization, device description via device class...); including OSI Layer 6 (Presentation). Middleware typically handles some session-level issues, but may utilize aspects of transport-layer session management when it is implemented over a transport such as TCP that already provides a form of session management.

⁴ We note that neither of the two key documents for smart grid interoperability [3, 20] even mention middleware by name, and the few examples [3] happens to give are very primitive and are mingled in with network protocols without distinguishing between the two.

In this vein, the Distribution Metering Standard, ANSI C12.22, is also aligned with the concept of middleware by providing services such as “naming” and “application security” if needed (Not provided by legacy lower layer communications technologies). It is also, constructed to operate above UDP and TCP transport [14] UDP, which is often used as a critical element for both more predictable low latency and natural fit within multicast frameworks, has no real session layer, so when middleware is implemented on top of it the middleware typically does all the session management (for example, when a client is “connected” to a server object in CORBA or a subscriber is “connected” to a publisher).

Further, we note that the GWAC layering perceptively includes a semantic understanding (interoperability) layer. There is as yet only limited experience with the appropriate ways to separate and integrate these newer views of interoperability, but it seems clear that middleware services can and should serve as a simplifying organizational base apart from the specific technology and mechanisms used to implement it, thus pushing our notion of “interoperable systems” to an even higher level.

Finally, we note that middleware is not mutually exclusive with existing utility standards such as C37.118, IEC 61850, ANSI C12.19/C12.22 and OPC UA. Indeed, middleware can be used to integrate these and other standards into a grid-wide inter-utility data delivery system within a standardized and hierarchical naming scheme. Additionally, some middleware is developed specifically for the wide area [21, 11]. Given that the above-mentioned utility standards feature protocols that have varied abilities for the wide area with QoS and multicast and security, such middleware can encapsulate and integrate with messages from current utility standards listed above, which may already have widespread deployment.

4. NON-FUNCTIONAL INTEROPERABILITY AND QOS STOVEPIPES

As noted above, functional behavior deals with the business logic embedded in programs. Its APIs are specified in an IDL or some similar kind of contract. While functional behavior deals with the “*what*” of the program or service, non-functional behavior deals with the “*how*”: how fast, how robust/available, how secure, how complete. Implementing this often also requires sophisticated *resource management strategies*, and encompasses specific issues that service providers are routinely concerned with, including such things as controlled sharing and utilization of resources.

4.1. Non-functional properties

Non-functional properties that are required for the power grid include the following [22, 23, 24, 25, 26, 27]:

- End-to-end latency (as low as a few milliseconds for current expected applications)
- Rate (from once a minute to 250 Hz)
- Widely varying requirements for availability of Data: {Ultra-high, Very High, Medium, Low }
- Confidentiality
- Integrity

A crucial point regarding non-functional properties is this—*you usually can't have them all at once*:

1. Different properties inherently must be traded off against others.
2. Different mechanisms for a given property are appropriate for only some of the operating conditions an application may encounter (especially a long-lived one).
3. Different mechanisms for the same non-functional property can have different tradeoffs of lower-level resources (CPU, bandwidth, storage)
4. Mechanisms most often can't be combined in arbitrary ways

Further, even if you somehow could have them all at once, it would likely be prohibitively expensive. Given these realities, and the fact application programmers rarely can be expert in dealing with the above issues, middleware with non-functional properties supported in a comprehensive and coherent way is a way to package up the handling of these issues and allow reuse across application families, organizations, and even industries. Indeed, for this reason, the Quality of Service for Objects project (QuO) middleware framework even has architecturally created a first-class role for a new kind of programmer: a QoS Engineer [11, 28, 29]

4.2. Implementing Non-Functional Properties

Resource allocation is a big part of resource management and is essential for providing non-functional properties. A given lower-level mechanism enables one or more non-functional properties that may be optimized (or, at minimum, appropriate) for some operating conditions and inappropriate or even considered “not working” under other conditions. At runtime, a given mechanism may utilize different levels of underlying resources (CPU, bandwidth, memory/storage). Different mechanisms providing the same property can provide different levels of non-functional

service for given operating conditions; they also typically offer different tradeoffs between the level of non-functional properties provided and resources consumed.

Examples of typical ways that non-functional properties can be supported include the following:

- Latency mechanisms: a chain of network-level “reservations” for performance (see below for a more detailed view).
- Confidentiality mechanisms: encryption
- Integrity mechanisms: higher-level algorithms built on top of encryption (e.g., digital signatures).
- Availability mechanisms: replication (spatial, temporal, value) and end-to-end latency mechanisms per above.

4.3. Abstraction Level for Non-Functional APIs

Best practices dictate that the abstraction level for non-functional properties offered to the programmer be established as high as possible, rather than encouraging developers to bind directly into lower-level mechanisms, for a number of reasons:

- It is less error-prone.
- Very few application programmers are expert in low-level, non-functional property mechanisms.
- Different lower-level mechanisms are available in different configurations in different deployments.
- The APIs of the lower-level mechanisms will change over time and perhaps with situation.
- New lower-level mechanisms providing the same property or properties will become available over the lifetime of an application (which often can span many decades). Such new mechanisms will often be better than existing ones in one or more ways, including offering a higher level of a non-functional property or being useable across a wider range of operating conditions

We now give an example of how higher-level properties can be mapped down to lower-level mechanisms [30]:

- Application-Level-1: **freshness** = $max_period + max_latency$
- Application-Level-2: **rate and latency** to deliver a given update over given path of links (each with given link-level latencies), for a given update message size. Note that the *max_period* above is inversely related to **rate** here.

- Network-Level-1: **bits/second** over a given link. This of course depends on the size of the updated variable (which may vary considerably) and the rate (which in some cases may be changed at runtime).
- Network-Level-2: **mechanism-specific parameters** of a given **network-level QoS mechanism**

In Section 5 below, we show the difficulties that application programmers may have in directly programming to these network-level QoS parameters without the added support from a middleware infrastructure layer.

4.4. QoS Stovepipe Systems

Recall from Section 2 the definition of a *stovepipe system*:

Stovepipe System: a legacy system that is an assemblage of inter-related elements that are so tightly bound together that the individual elements cannot be differentiated, upgraded, or refactored. The stovepipe system must be maintained until it can be entirely replaced by a new system [12, 13]

From this we propose the following new definition:

QoS Stovepipe System (QSS): a system of systems whose subsystems are locked into low-level mechanisms for QoS and security such that

- a) it cannot be deployed in many reasonable configurations, or
- b) some programs cannot be combined because they use different lower-level QoS mechanisms for the same property (e.g., latency) that cannot be composed, or
- c) It cannot be upgraded to “ride the technology” curve as better low-level QoS and security mechanisms become available.

It is essential that any “smart grid” avoids enabling or perhaps even allowing QSS, and in the next section we discuss how middleware can help.

5. MIDDLEWARE AND QOS INTEROPERABILITY

Common network-level QoS mechanisms include ATM, INTSERV/RSVP, IPv6 Flow Labels, DIFFSERV, and MPLS. These different mechanisms all have service level management capabilities, very roughly parameterized by delay, loss, throughput, and security. However, they have very different semantics. Most offer very coarse notions of these properties, although IPv6 Flow Labels [31] offer somewhat finer granularity (though likely not nearly what is needed for real-time streaming of mission critical data such as represented by synchrophasor based applications).

Protocol	Guarantees	Control Mechanism	Control Time
ATM	Strong	Reserve explicit circuit across network for entire connection	Runtime: connection Setup; or SLA purchase time
INTSERV/RSVP	Soft (will inform program if violated)	Out-of-band channel following data path, setting up time slots or buffers etc.	Runtime: out of band, can be repeated
IPv6 Flow Label	Hints (not even soft), will likely happen in the aggregate over a long time	Byte in each IP packet tagged with a class; can be interpreted differently in different implementations	SLA purchase time
DIFFSERV	Hints (not even soft), will likely happen in the aggregate over a long time	Byte in each IP packet tagged with a class; can be interpreted differently in different implementations	SLA purchase time
MPLS (specialized form of DIFFSERV)	Aggregate economic guarantees over {user, location, protocol}	Internal to ISP: in ingress ISP adds header tag and used internally to queue by class and user, not packet	Traffic shaping time: out of band, periodically, aggregated across multiple customers.

Figure 3: Differences in Network-Level Mechanisms for {delay, loss, bandwidth, security}

But these different mechanisms that seem superficially similar vary quite a bit in terms of the sustainable service they provide, the kind of control mechanism (and corresponding API), and the time that this control mechanism is invoked by the program. Figure 3 summarizes how ATM, INTSERV/RSVP, IPv6 Flow Labels, DIFFSERV, and MPLS all vary widely in such ways. It is worth noting regarding these different mechanisms:

- Composing them (e.g., across ISPs or organizations) is not a simple task
- None are likely to become the single standard or protocol and be available everywhere, so composition or augmentation may be necessary.

These together argue that it is best to not burden application programmers with individually directly programming to these mechanisms for supporting the associated non-functional properties. Rather, their use can be incorporated as best practices into middleware, with common mappings to one or more underlying mechanisms of choice. This is essential if we are to avoid building QSSs for the “smart grid”. In our long experience in building distributed applications, it is very difficult to avoid QoS Stovepipe Systems without an interceding layer of QoS-enabled middleware. Such QoS-enabled middleware can be provided by experts through common infrastructure not only for functional interoperability but also for non-functional properties; such middleware has been under research study and transition evaluation since the mid- 1990s and later has

been offered as standardized commercial products (e.g., by the OMG) [32]. This allows middleware vendors to establish which mechanisms providing different non-functional properties can be used and combined in which ways under which operating conditions, and package this up for programmers to use at a higher level. For research examples of such middleware, see [33] [34] [35]. Further commentary on this issue can be found at [36].

Finally, regarding middleware, in our opinion you cannot today buy everything that is needed for complex high performance, high precision and highly predictable mission critical data delivery systems commercial off the shelf (COTS), especially for wide area deployments. Some existing COTS middleware can be very appropriate for significant parts of such systems. However, these implementations are not optimized for the very low latencies and the very high availabilities that wide-area system integrity protection schemes [37] and closed-loop control for the grid will require. We note that besides QoS-enabled middleware, of course, there still needs to be a coherent architecture that utilizes it [38], and a middleware based perspective can be an important element of that as well.

Acknowledgements

We received quite useful and detailed feedback on this paper from John Zinky, Carl Hauser, and Adam Bakken. We also thank Jeff Dagle, Stan Schneider, Anjan Bose, Phil Overholt, Hank Kenchington, Neeraj Suri, and Alison

Silverstein for their inputs to the research that lead to this paper.

This research has been supported in part by grants CNS 05-24695 (CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)) and its new extension, TCIPG. All above have been funded by NSF, DHS, and DoE, and for that we are grateful.

References

- [1] GridWise Architecture Council, [Interoperability Path Foreward Whitepaper](#), November 2005.
- [2] GridWise Architecture Council, [Interoperability Constitution Whitepaper \(v1.1\)](#), December 2006.
- [3] GridWise Architecture Council, [GridWise Interoperability Context-Setting Framework \(v1.1\)](#), March 2008.
- [4] C. Hauser, D. Bakken, and A. Bose, [A Failure to Communicate: Next Generation Communication Requirements, Technologies, and Architecture for the Electric Power Grid](#), IEEE Power & Energy Magazine, 3(2), March 2005, 47-55.
- [5] K. Tomsovic, D. Bakken, M. Venkatasubramanian, and A. Bose, [Designing the Next Generation of Real-Time Control, Communication and Computations for Large Power Systems](#). In *Proceedings of the IEEE* (Special Issue on Energy Infrastructure Systems), page 93(5), May 2005.
- [6] North American Synchrophasor Initiative, [Quanta Statement of Work](#), May 2008.
- [7] D. Bakken, [Middleware](#), a survey article prepared for *Encyclopedia of Distributed Computing*, Kluwer Academic Publishers, Partha Dasgupta and Joseph Urban, editors.
- [8] S. Schneider, [Middleware for Mission-Critical Systems](#), presentation at NASPI Working Group Meeting, October 7, 2009, Chattanooga, TN.
- [9] R. Soley. [Open Standards for Middleware: and Up the Stack Too](#), presentation at NASPI Working Group Meeting, October 8, 2009, Chattanooga, TN.
- [10] R. Schantz. [Quality of Service](#), a survey article prepared for *Encyclopedia of Distributed Computing*, Kluwer Academic Publishers, P. Dasgupta and J. Urban, editors.
- [11] Zinky JA, Bakken DE, Schantz R. [Architectural Support for Quality of Service for CORBA Objects](#). Theory and Practice of Object Systems, April 1997.
- [12] Wikipedia, [Stovepipe system](#).
- [13] [Improving Project Management in the Department of Energy](#), National Academy Press, ISBN 0-309-06626-3, 1999.
- [14] IETF RFC C1222 Transport Over IP – 2009.
- [15] Geihs, K., "[Middleware challenges ahead](#)," *Computer*, vol.34, no.6, pp.24-31, Jun 2001.
- [16] Richard E. Schantz, Robert H. Thomas, Girome Bono. "The Architecture of the Cronus Distributed Operating System". In *Proceedings of the 6th International Conference on Distributed Computing Systems (ICDCS86)*, IEEE Computer Society Press, Cambridge, Massachusetts, USA, May 19-13, 1986, 250-259.
- [17] Richard E. Schantz, Joseph Loyall, Craig Rodrigues, Douglas C. Schmidt, Yamuna Krishnamurthy, and Irfan Pyarali. [Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware](#). The ACM/IFIP/USENIX International Middleware Conference, June 2003, Rio de Janeiro, Brazil.
- [18] Richard E. Schantz and Douglas C. Schmidt. [Research Advances in Middleware for Distributed Systems: State of the Art](#). IFIP World Computer Congress, August 2002, Montreal, Canada.
- [19] Richard Schantz and Doug Schmidt. [Middleware for Distributed Systems - Evolving the Common Structure for Network-centric Applications](#). Chapter in the Encyclopedia of Software Engineering. John Wiley & Sons. December 2001, pp. 801-813.
- [20] [Framework and Roadmap for Smart Grid Interoperability Standards Release 1.0 \(Draft\)](#), NIST, September 2009.
- [21] D. Bakken, C. Hauser, H. Gjermundrød, and A. Bose. [Towards More Flexible and Robust Data Delivery for Monitoring and Control of the Electric Power Grid](#), Technical Report EECS-GS-009, Washington State University, May 2007.
- [22] *IEEE 1646 Standard Communication Delivery Time Performance Requirements for Electric Power Substation Automation*, IEEE, 2004.

- [23] David Bakken, [Quality of Service Design Considerations for NASPInet](#), presentation at NASPI Working Group Meeting, February 4, 2009, Scottsdale, AZ.
- [24] IntelliGrid Project, "The Integrated Energy and Communication Systems Architecture, Vol. IV: Technical Analysis".2004, Available via <http://www.epri.com/IntelliGrid/>.
- [25] US Dept. of Energy, [Roadmap to Secure Control Systems in the Energy Sector](#), January 2006.
- [26] K. Hopkinson, G. Roberts, X. Wang, and J. Thorp, [Quality of Service Considerations in Utility Communication Networks](#), IEEE Transactions on Power Delivery, 24(3), July 2009.
- [27] Dave Bakken, Carl Hauser, and Harald Gjermundrød, [Appropriateness of internet protocols and commercial publish-subscribe middleware for wide-area data delivery in the bulk power system](#). Technical Report EECS-GS-014, Washington State University, November, 2009.
- [28] Schantz RE, Loyall JP, Atighetchi M, Pal PP. [Packaging Quality of Service Control Behaviors for Reuse](#). Proceedings of ISORC 2002, The 5th IEEE International Symposium on Object-Oriented Real-time distributed Computing, April 29 - May 1, 2002, Washington, DC.
- [29] Pal PP, Loyall JP, Schantz RE, Zinky JA, Shapiro R, Megquier J. [Using QDL to Specify QoS Aware Distributed \(QuO\) Application Configuration](#). Proceedings of ISORC 2000, In *Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-time distributed Computing*, March 15 - 17, 2000, Newport Beach, CA.
- [30] David Bakken. [Quality of Service Design Considerations for NASPInet](#). Presentation to the North American Synchrophasor Initiative (NASPI) Work Group meeting, Scottsdale, AZ February 4, 2009.
- [31] Sun Microsystems, [IPv6 Quality of Service Capabilities](#), 2009.
- [32] Krishnamurthy Y, Kachroo V, Karr DA, Rodrigues C, Loyall JP, Schantz RE, Schmidt DC. [Integration of QoS-Enabled Distributed Object Computing Middleware for Developing Next-Generation Distributed Applications](#). In *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*, June 18, 2001, Snowbird, Utah.
- [33] Christopher Gill, Jeanna Gossett, David Corman, Joseph Loyall, Richard Schantz, Michael Atighetchi, and Douglas Schmidt. [Integrated Adaptive QoS Management in Middleware: A Case Study](#). 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004), Toronto, Canada, May 25-28, 2004.
- [34] Loyall JP, Bakken DE, Schantz RE, Zinky JA, Karr DA, Vanegas R, Anderson KR. [QoS Aspect Languages and Their Runtime Integration](#). *Lecture Notes in Computer Science*, Vol. 1511, Springer-Verlag. Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), 28-30 May 1998, Pittsburgh, Pennsylvania.
- [35] Richard Schantz, [Evolution of Middleware Services toward Realtime and Embedded \(Cyber-Physical\) Environments: The BBN Experience](#), presentation at NASPI Working Group Meeting, February 5, 2009, Scottsdale, AZ.
- [36] L. Beard, D. Bakken, F. Galvan, and P. Overholt, [Data Delivery & Interoperability for Smart Grids](#), presentation at Grid-Interop 2008, November 11-13, 2008, Atlanta, GA.
- [37] Horowitz, S. Novosel, D. Madani, V. Adamiak, M. "[System-Wide Protection](#)", *IEEE Power & Energy Magazine*, 6(5), September 2008, 34-42.
- [38] R. Tucker, [End to End communications for Smart Grid](#), Technical Report, North American End Device Registration Authority (NAEDRA), April 16, 2009.

Biography

Dr. David E. Bakken is an Associate Professor of Computer Science at Washington State University. His expertise includes designing, implementing, and deploying middleware frameworks supporting multiple QoS/security properties for wide-area networks. He has been working closely with WSU power researchers for 10 years on rethinking the grid's limited communications and developing the GridStat middleware framework. GridStat has had a large impact on the shape of the emerging NASPInet framework, which is the leading effort to develop better wide-area data delivery for the bulk power system. Prior to joining WSU, he was a scientist at BBN Technologies where he was an original co-inventor of the Quality Objects (QuO) framework. He has consulted for Amazon.com, Network Associates Labs (formerly Trusted Information Systems), and others; and he has also worked as a software developer for Boeing.

Dr. Richard E. Schantz is a principal scientist at BBN Technologies in Cambridge, Mass., where he has been a key contributor to advanced distributed computing technology and transition to common practice for the past 35 years. His research has been instrumental in defining and evolving the concepts underlying middleware since its emergence in

the early days of the Internet. He was directly responsible for developing the first operational distributed object computing capability and transitioning it to production use. Recently as principal investigator on a number of key DARPA projects in the areas of adaptive realtime behavior, system survivability and advanced software engineering, he has led research efforts toward developing and demonstrating within network-centric applications the effectiveness of a new generation of middleware support for adaptively managing Quality of Service control.

Richard D. Tucker, PE, is a professional engineer experienced in distribution metering and instrumentation. The interoperability quest was a defensive mechanism to deal with the multitude of proprietary metering communications protocols. Working with Measurement Canada, ANSI and IEEE from 1992 till 2009, the three Standards bodies created ANSI C12.18, C12.21, C12.19 and C12.22 for North American Utility metering and instrumentation. He has held Chair position IEEE End Device/TIU subcommittee since 1992. He was honored during the 2009 SANTA CLARA, CA Connectivity Week as one of the Leaders in smart grid interoperability by GridWise Architecture Council (GWAC) for advancing openness in the smart electric power system of the future.