WHITEPAPER

Accelerating the Industrial Internet with the OMG Data Distribution Service

Dr. Douglas C. Schmidt, Professor of Computer Science, Vanderbilt University

Abstract

The Industrial Internet is an emerging software and communication infrastructure that interconnects machines and data to build intelligent machines and applications never before possible. Using GE's words, it merges "big iron" and "big data" to create "brilliant machines." Embedded sensors and sophisticated software allow machines (and users) to communicate in real-time and find meaning where it did not exist before. Machines—from jet engines to gas turbines to medical scanners—connected via the Industrial Internet have the analytical intelligence to self-diagnose and self-correct, so they can operate reliably, react to real-world changes in their environment and provide more sophisticated service to the users.

This paper describes advances taking place in the Industrial Internet. The content will cover technical challenges emerging in this context, including terminology, history and research. It also showcases the OMG Data Distribution Service (DDS) as a critical foundation for building an elastic software infrastructure for the Industrial Internet.

Overview of the Industrial Internet

The term Industrial Internet was coined by GE several years ago and refers to the integration of complex physical machinery with network sensors and software. The concept combines several technologies within the field, including machine learning, big data, Internet of Things (IoT) and machine-to-machine (M2M) communication.

The goal of the Industrial Internet is to facilitate the connection of machines embedded with sensors to other machines and end-users, as well as to enable access and control of various mechanical devices. GE often refers to mechanical devices as "things that spin," which includes turbines, jet engines, and medical devices. The key to the success of these systems is their ability to extract data from devices, make sense of that data in real time and deliver it at a specific time. The end goal is to derive value in terms of improved utility and cost savings.

Overview of Cyber-Physical Systems (CPSs)

At the heart of the Industrial Internet are cyber-physical systems (CPSs) operating within cloud computing environments. CPSs involve tight coupling and coordination between a system's computational elements, components written in software and physical elements, which are components that interact with the physical world. Examples of physical elements include hot rolling mills, medical instruments, airplanes, trains and anti-lock breaking systems on modern automobiles (see Fig. 1).

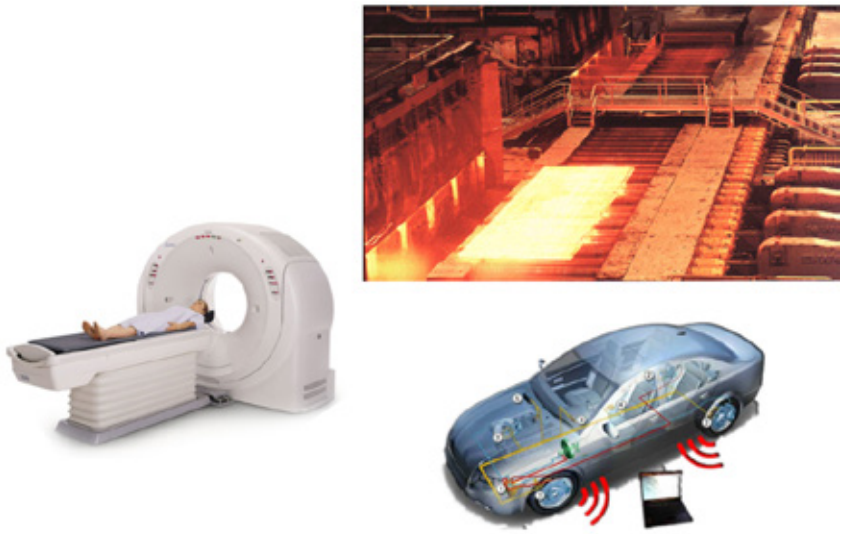

Figure 1. Cyber-physical systems (CPSs) feature tight coordination between computational and physical elements.

CPSs increasingly use network processing elements to control devices and interactions. Interactions may include physical environments, such as wind farms, chemical environments or biological environments, such as pharmaceutical factories. These interactions may also be used for experiments in big data, big science, smart grid power control and other domains (see Fig. 2).

Figure 2. CPSs increasingly use networked processing elements to control physical, chemical or biological processes or devices.

CPSs are time-sensitive, meaning that correct information or action delivered or executed too late results in an incorrect outcome. As a result, the dependability of CPSs has not only a reliability dimension but also a temporal one. In other words, system functions must occur in a timely manner. Security is increasingly important to CPSs since it is detrimental to deliver information at a specific time if the information has been tampered with or compromised.

After decades of research and development on these CPSs in industrial, military, aerospace, and telecom, one particularly illustrative example of a mission-critical CPS comes from a company in Germany that builds quality control equipment for beer bottling. This system takes photos of beer bottles during the manufacturing process and performs image processing to monitor quality. Image processing of the photos is used to detect defects in the glass, mold in the beer and other imperfections.

Maximized Data to Packet Ratio

The DDS wire protocol was designed for distributed real-time applications. It was therefore originally designed to be highly efficient on the wire. For example, data type information is exchanged once at discovery time, not continuously at run-time. Also, the open OMG standard wire protocol of DDS (called the DDS-RTPS Interoperability Wire Protocol) uses a very compact and efficient binary wire data representation (Common Data Representation, or CDR). RTI has numerous benchmarks that support its throughput capabilities (see http://www.rti.com/products/dds/benchmarks-cpp-linux.html).

The application developer is also given fine-grain control of packet sizes, which is particularly important in low bandwidth and high latency networks where application data needs to be placed into large data packets to minimize the impact of latency on the application. RTI Data Distribution Service also uses type information intelligently to put minimum data on the wire. For example, it supports sparse data types so that for a given type, only changed fields would be transmitted on the wire (see http://www.rti.com/products/index.html).

Distributed Cyber-Physical Systems and the Role of Cloud Computing

CPSs have historically been used to control devices in stand-alone environments that often aren't connected to a network at all. Although there are still many of these types of CPSs, this paper focuses on CPSs that are connected via local area networks (LANs) or wide area networks (WANs). These distributed CPSs must address many requirements and challenges that aren't as relevant for stand-alone CPSs, including partial failure and higher latency and jitter due to shared communication links, as well as denial of service attacks.

Although there are a number of examples of distributed CPSs (primarily in the aerospace and defense domains), these types of systems have historically been highly proprietary and expensive to develop and sustain. In recent years, the enormous investment in commodity cloud computing environments has spurred an interest in leveraging these technologies as the basis for distributed CPSs. Cloud computing provides a utility model for computing that enables ubiquitous, convenient and on-demand access across a network to a shared pool of configurable computing resources.

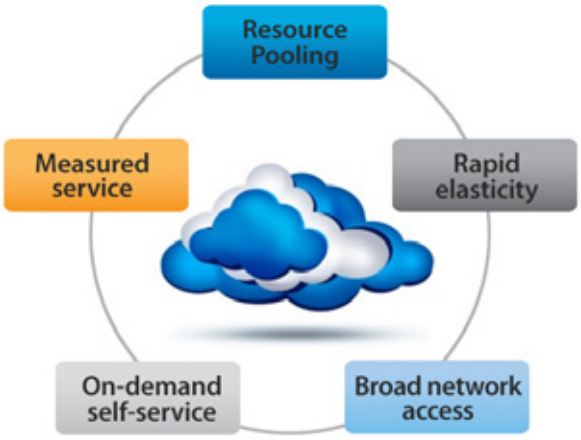

Figure 3. Key characteristics of cloud computing environments.

The key characteristics of cloud computing environments (see Fig. 3) include:

- **On-demand self-service**, which refers to the ability of consumers to unilaterally provision computing capabilities including networks, storage and servers. This requires the ability to interact over high-speed broadband networks.
- **Resource pooling and multi-tenant models,** in which multiple applications can run in the context of shared server and networking resources. Achieving these elastic capabilities requires the means to automatically and rapidly expand and contract the amount of computation and storage based on dynamically fluctuating levels of demand.
- **Measured service,** in which resource utilization can be controlled via some type of metering capability.

The goal of cloud computing is to treat computing and communications as utilities and provide these capabilities without requiring excessive involvement from operators or service providers, as it should instead be achieved largely through application control. A key benefit of cloud computing lies in the economies of scale provided by multi-tenancy and elasticity, which involve the ability to have multiple applications and services sharing the same computing infrastructure, as well as the potential to expand and contract infrastructure as needed and on-demand. Virtualization can be useful in these scenarios, but it may become detrimental in cyber-physical systems due to overhead and scheduling issues.

Most applications of commodity cloud computing focus on web hosting, where low cost (e.g., via resource sharing) and high availability (e.g., via replication) are the most critical quality-of-service (QoS) properties. Although these environments are increasingly being adopted by consumers and certain industries, many of the classic implementations of cloud computing are at odds with CPS requirements, such as bounded latency, jitter and priority inversions.

Research and Development Progress for Distributed Cyber-Physical Systems

Many CPS design and operational paradigms have come and gone during the past 40 years. In the 1970s and 1980s, there was a tendency to build CPSs via a tightly-coupled design paradigm (see Fig. 4).

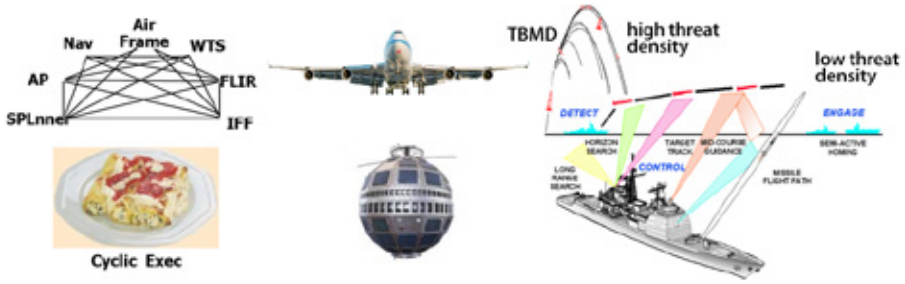

Figure 4. Legacy CPS designs tend to be stovepiped, propietary, brittle and nonadaptive, vulnerable and expensive to develop and evolve.

These CPSs were designed in a stovepipe manner with many and little reuse or sharing. Most elements of these systems were proprietary and controlled or built by a single integrator. They were also non-adaptive. For example, if changes were made to requirements or the runtime environment, many parts of the systems broke. As a result, these CPSs were expensive to sustain and evolve, in addition to incurring vulnerabilities due to not being designed to connect to the Internet.

In general, the limitations of the tightly-coupled design paradigm are that small changes made to the system can break almost anything. Examples of these problematic changes include adjustments to requirements, implementation, infrastructure, operating systems, programming languages, middleware and networks.

This tightly-coupled design paradigm is also problematic due to the ways in which developers and operators have traditionally provisioned, scheduled and certified CPSs. The operational capabilities and characteristics of traditional CPSs are typified by the need to obtain all the required resources. If this provisioning process goes smoothly, traditional CPSs usually work well. If not all of the resources are acquired, however, there can be major issues and CPSs simply do not work as needed (see Fig. 5).

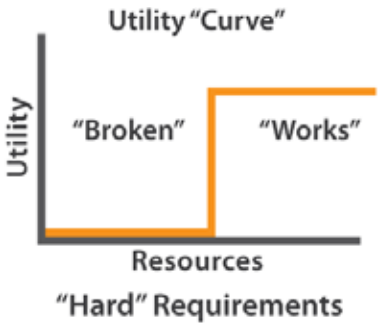

Figure 5. Tightly-coupled design paradigm.

The tight coupling of these CPSs is exacerbated by stringent end-to-end QoS requirements, including bounded latency and absence of priority inversion. To meet these requirements, developers of traditional CPSs typically lock everything down, share limited information between priority groups and allocate resources statically. While this strategy works for small CPSs in closed stand-alone environments, it simply doesn't scale to meet the needs of distributed CPSs being developed and planned for today's Industrial Internet. In particular, computing clouds aren't typically viewed as platforms that must be statically provisioned or tightly managed without sharing.

Over the past decade, there has been tremendous improvement in research and development for distributed CPSs, as well as evolution in the adoption and application of newer design paradigms. For example, cutting-edge CPSs in military, aviation and civilian domains are more layered and componentized than those of previous decades (see Fig. 6).

Figure 6. Modern, leading-edge CPS designs tend to be layered and componentized, standards and COTS-based, robust to failures, adaptive to operating conditions and cost effective to evolve and retarget.

Modern CPSs include layers of standardization and have become more robust at the infrastructure level. Moreover, advances in loosely-coupled CPS software and system architecture have improved such that when problems arise, properly programmed systems are able to cope using appropriate adaptive modifications.

A further benefit of these modern, less tightly-coupled CPSs is that solutions are more cost-effective to evolve and retarget. Developers are less apt to have to backtrack and recertify an entire CPS when minor changes are made, which is a key cost-driver for sustainability in legacy systems. Consequently, changes can be made to a CPS environment, requirements and aspects of implementation, including those that are hidden behind component or module boundaries.

Modern CPSs have improved from an operational point of view. The majority of new loosely-coupled CPSs are being constructed via data-centric and reusable protocols. Event and messaging buses are more resilient in these types of CPSs. The CPSs themselves, when constructed properly, are designed to work appropriately even if they don't receive all resources in a timely manner, which enables dynamic allocation and management (see Fig. 7). There is the added benefit of better sharing support for resources, especially in environments with the ability to describe priorities and importance of information flow at multiple levels.

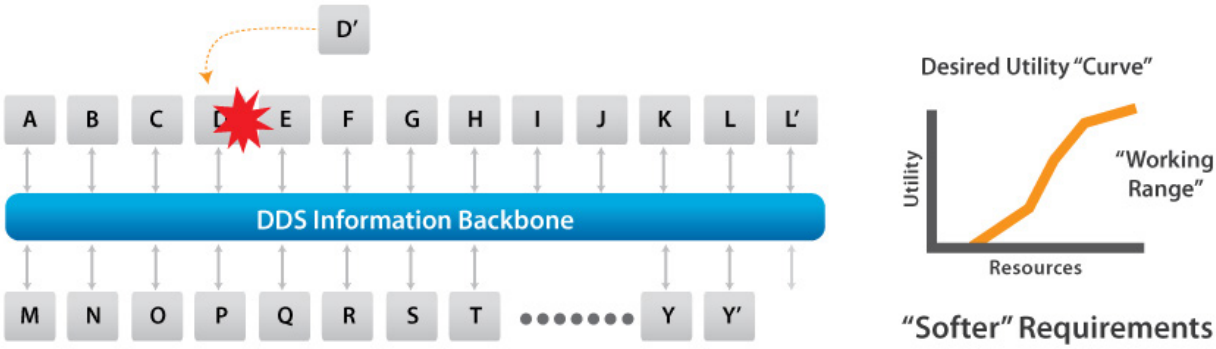

Figure 7. Loosely-coupled CPSs ensure acceptable end-to-end QoS and enable dynamic resource allocation and management.

Some of the operating platforms that have evolved to support modern CPSs have much in common with computing clouds. For example, the total ship computing environment developed for the US Navy's DDG-1000 destroyer include advances in distributed resource management based on many of the technologies mentioned throughout this paper (also see http://www.dre.vanderbilt.edu/~schmidt/JSS-DRM.pdf). While the scale of a DDG-1000 destroyer is not nearly as large as the Industrial Internet, it serves as a good example of how metropolitan area network (MAN)-sized CPSs are being developed reliably and securely.

Current Research and Development Trends and Challenges for Distributed Cyber-Physical Systems

Current trends and challenges within the domain of distributed CPSs are a hot topic of discussion. Considering the activity in this space, it is useful to be familiar with past developer approaches and gain insight from those experiences.

Elastic Hardware

The CPS space is complicated, but some answers can be found in research conducted on elastic hardware platforms in cloud computing environments. Elastic hardware refers to platforms with the ability to add or remove CPU capacity within a reasonable time frame and price. This technology enables cloud providers to add or subtract hardware without the need to change underlying business logic or software configurations. Since programmers' time has become a precious commodity, the flexibility enabled by elastic hardware is tremendously valuable.

One complication of elastic hardware is that most platforms have been utilized for hosting web applications in public cloud environments or data centers. Although those environments have been relatively reliable for conventional web hosting services, they pale in comparison to the complexities and mission-criticalities of Industrial-Internet-style applications, where support for secure, real-time communications and failover are essential.

Elastic hardware is thus necessary, but not sufficient, for building elastic applications that possess cyber-physical properties. There are a number of reasons why programming elastic hardware for CPS is hard. The first is that many programming models used by developers are inadequate. Developers tend to use complicated or obtrusive APIs, which are challenging to program. Conversely, there are solutions that are simple to program, but tend to have problems with respect to scalability and predictability. These solutions work well if timeliness is not a concern, but they are not a viable solution when timeliness is paramount.

Another issue is the general lack of understanding for real-time, concurrent network solutions. There are many inherent and accidental complexities in this area, including race conditions, deadlocks, priority inversions and missed deadlines. The CPS development community needs to become more familiar with these issues so they can work more effectively at fixing them with available tools.

Some operating platforms provide good support for multicore solutions, but do not have sufficient support to seamlessly transition from multicore to distributed core. When this is the case, the system will work well up to about 16 cores (i.e., the current scale supported by high-end Intel or AMD multicore chip sets), and then start to degrade significantly when the system scales beyond that.

Finally, there is the long observed issue of inadequate support for QoS at scale. In this context, QoS refers to the ability to control systematic quality attributes (often referred to as "non-functional properties"), including prioritization, failover and robustness, as well as system-wide resources in an end-to-end environment over various types of networking infrastructure. Approaches that work well for conventional web-based systems often do not work as well in the mission-critical CPS domain.

The impediments to programming elastic applications on elastic hardware affect the majority of computing systems, though they are particularly problematic for CPSs. As a result, many organizations believe at their peril that the traditional Internet and Web protocols that work well for ecommerce or file sharing will work just as well for more complex CPSs in the Industrial Internet. When they ultimately discover otherwise, the consequences are expensive, at best.

Four Key Research Challenges for Elastic Cyber-Physical Systems to Support the Industrial Internet

As discussed at the beginning of this paper, the Industrial Internet is an emerging software and communication infrastructure that connects machines, data and their users, to enable access and control of mechanical devices in unprecedented ways. It connects machines embedded with sensors and sophisticated software to other machines (and end users) to extract data, make sense of it and find meaning where it did not exist before. Machines—from jet engines to gas turbines to medical scanners—connected via the Industrial Internet have the analytical intelligence to self-diagnose and self-correct so they can deliver the right information to the right people at the right time (and in real-time).

Despite the promise of the Industrial Internet, however, supporting the end-to-end QoS requirements is hard and requires advances in a number of key areas, including:

- **Precise auto scaling of resources** with an end-to-end focus needs to be a feature of CPS. Auto scaling is often thought about as adding cores when demand rises and removing cores when demand wanes (see Fig. 8). Although this is certainly useful, it comes with the downside of not working properly from a system-wide perspective. Large-scale systems like the Industrial Internet require ways to scale up scheduling and auto scaling in a broad environment, to support precise behavior for end-to-end task changes.

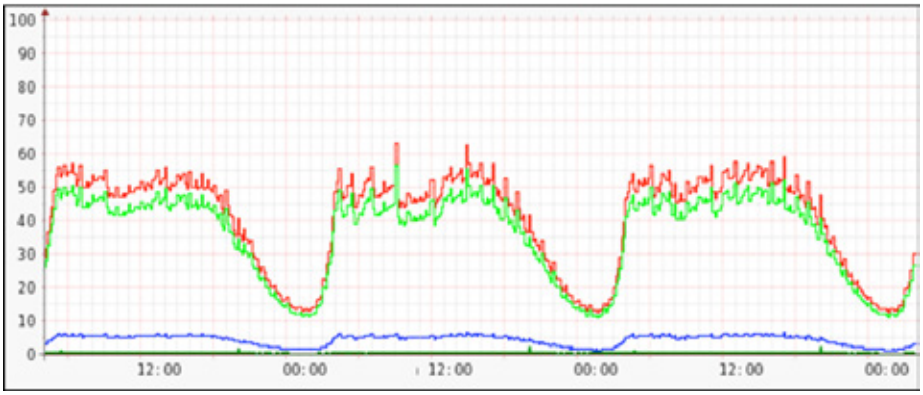

Figure 8. CPU utilization.

Stability and safety properties within mission-critical CPSs require complex analysis to provide confidence that they will work as expected. Supporting this need calls for analysis examining reachability of states in system, which is currently a particularly hard part of the research space.

- **Optimization algorithms that balance real-time constraints with cost** and other goals must be in place (see Fig. 9). These problems can be solved by additional hardware, but not all developers have those resources available. Although deployment and configuration algorithms—along with services and infrastructure—are key to successful CPSs, implementing these algorithms effectively is hard in domains where the cost commodity marginal basis is driven down. For example, the automotive industry needs to sell in volume, and thus cannot afford to spend hundreds of dollars on high-end hardware in low-end to mid-level cars because the costs will not be recouped.

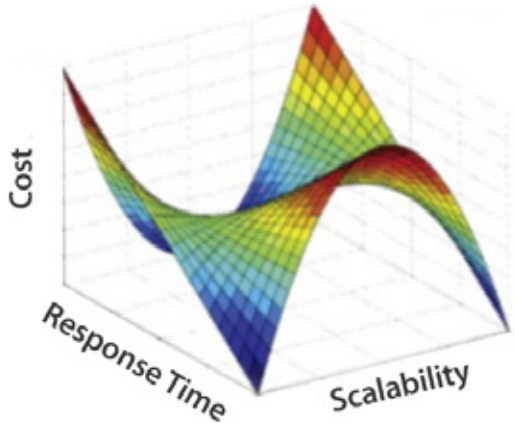

Figure 9. Multidimensional resource management.

Another essential component for CPSs is creating the means to co-schedule or perform admission control and eviction of assorted task sets deployed on shared computing and communication resources to ensure that high priority operations take place at the appropriate time. These requirements are not typically met in conventional cloud computing environments. In other words, when these systems get overloaded, the QoS degrades and there is no clear way to prioritize between tasks.

- Improved fault-tolerance fail-over that supports real-time requirements is crucial in environments with high probability of failures and attacks. One way to do this is semi-active replication, which is used so that when the system is running and failures occur, they can failover rapidly and predictably (see Fig. 10).

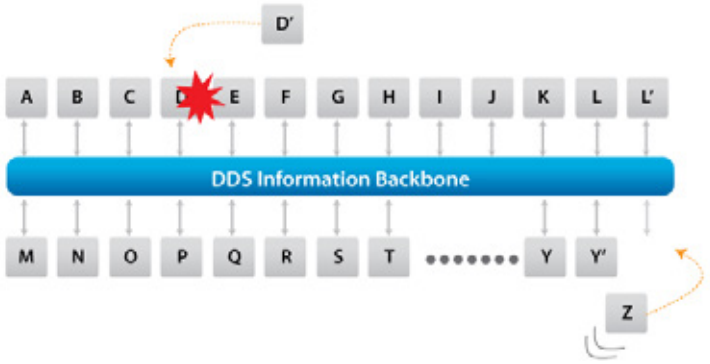

Figure 10. Semi-active replication is used for rapid and predictable failover.

This replication style is designed to have some of the benefits of both the active replication and passive replication styles, including predictable failover times and deterministic behavior during program execution. (see http://www.dre.vanderbilt.edu/~schmidt/PDF/WDMS02.pdf)

- **Data provisioning and load balancing algorithms that take into account a variety of properties**, including geo-physical, when deciding where to migrate work (see Fig. 11). Cloud computing is generally considered as so flexible that there is little difference to where computation takes place and storage resides, which makes sense when there are no real-time QoS needs. As real-time QoS needs arise, the location where parts of the system will run becomes more important. In these cases, affinity should be emphasized to reduce latency and jitter.

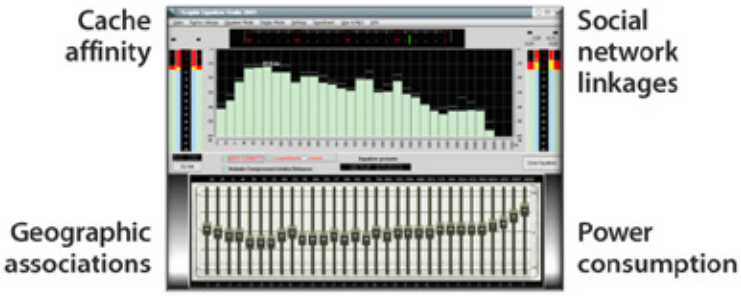

Figure 11. Data provisioning and load balancing algorithms use many different properties when deciding where to migrate work.

Storage is a key factor in CPS, as it does not do much good to virtualize storage if it then takes too long to move data from one node to another. At the same time, rebalancing and replication also need to happen. Taking physical dimensions into account in the context of load building is beneficial and not practiced as often as it needs to be. Developers must also discover a way to exploit physical characteristics of data and computation to better distribute work throughout clouds.

Developers of distributed CPSs need a holistic approach. Advance in this area is challenging because many researchers work in isolation, while most product companies work on projects one or two layers at a time. Success requires approaches from both research and product points of view that span the layers of these projects and can work end-to-end.

A Promising Solution

Meeting the challenges of distributed CPSs—including, but not limited to, the Industrial Internet—requires rethinking basic properties and principles commonly ascribed to cloud computing. Whatever the final solution for elastic cyber-physical systems software infrastructure may be, it must include support for the following requirements:

- Systems must be flexible, as they must be able to replace, reuse, analyze, distribute, paralyze in isolation and then compose these pieces back together in a dependable way.
- Systems need to be open so that programmers do not program themselves into a corner with a solution that only works with commitment to a single vendor.
- Systems need to be uniform with respect to treating multicore and distributed core in a common way. Uniformity keeps these two components transparent from the applications and services they run.
- Systems must be scalable as the demand for ever-increasing scope rises. Solutions such as load balancing algorithms take advantage of elastic hardware resources at the infrastructure level.

One of the most important considerations for meeting these requirements of distributed CPS is middleware, which resides between applications and the underlying operating systems, networks and hardware. Middleware provides key services that are essential to design and operate distributed CPSs at the scale of the Industrial Internet and ultra-large-scale (ULS) systems.

Key Layers of Distributed CPS Software Infrastructure

Anyone who has taken a networking course knows that there are seven layers in the OSI stack and four layers in the Internet stack. In general, people are less familiar with the layers within the middleware stack, which is essential to successfully developing next-generation software infrastructure for distributed CPSs and the Industrial Internet. Figure 12 illustrates the key layers.

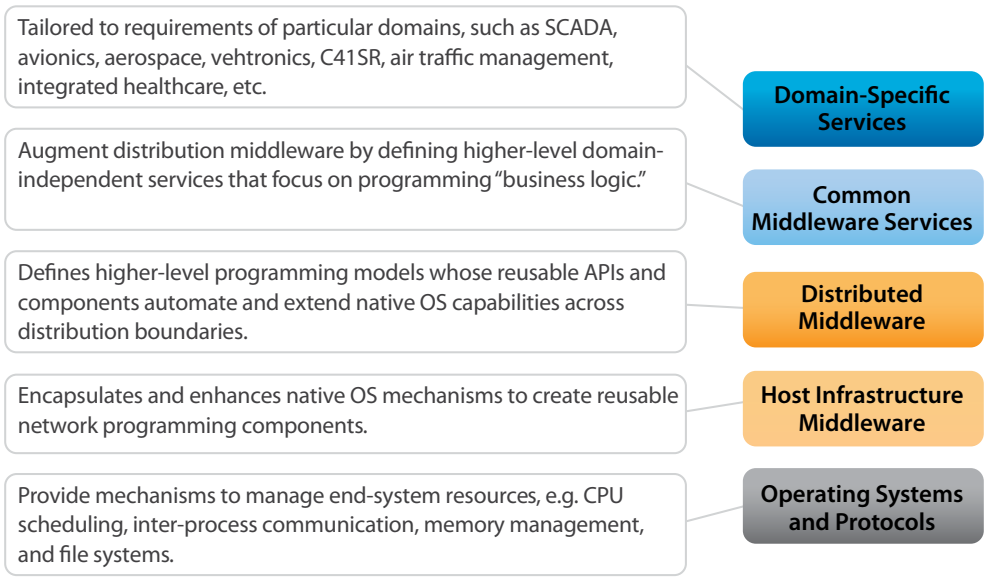

Figure 12. Key layers of distributed CPS software infrastructure.

An **operating system** is a hardware abstraction layer that allows higher-level services and applications to ignore differences in the hardware. Examples of operating sytsems for CPSs include real-time Linux, VxWorks and Solaris. Host infrastructure middleware is an operating system abstraction later that abstracts away from the operating system and removes accidental complexities of the system's APIs. It amplifies programming software in a portable way. Examples of host infrastructure middleware for CPSs include real-time Java and ACE.

The next level up is **distribution middleware**, which allows for decoupling and abstracting the fact that there is a network between the sender and receiver of messages. Distribution middleware provides the ability to communicate across address and host boundaries in a way that is unobtrusive to the application. Examples of this type of middleware for CPSs include real-time CORBA and DDS.

Common middleware services comprise the next layer. Once distribution middleware is implemented, it becomes easier to program across a network. The next challenge is deciding how to build reusable services that name the information, discover services, detect presence, send events to subscribers in a predictable way, monitor health, provide information durability and historical data, record data flows and transactions, perform failover operations, and so on. These all fall within the realm of common middleware services.

Domain-specific middleware services are perhaps the most important layer. These middleware services involve intellectual property or value added in a particular domain such as avionics, SCADA, C4ISR, air traffic management and healthcare. This area is where the bulk of the industrial Internet lies, and where the next generation of standards and capabilities must be researched.

Promising Elastic CPS Middleware: The OMG Data Distribution Service (DDS)

The Object Management Group (OMG) Data Distribution Service (DDS) for Real-Time Systems possesses all of the criteria for distributed CPS software infrastructure mentioned above. It is flexible, open, uniform and scalable. DDS supports a pattern language that allows loosely coupled, heterogeneous, evolvable, scalable and dependable distributed CPS. DDS is used widely throughout this domain because it provides a powerful software infrastructure for building Industrial Internet and other large-scale CPSs.

DDS supports different types of information modeling, including relational. Relational modeling uses a data-centric publish-subscribe abstraction in which events and their relationships to each other may be assigned. DDS reinforces the idea of a global data space, which enables the ability to read and write data asynchronously, anonymously and decoupled in time and space (see Fig. 13). It allows production and consumption of data in the global data space in many ways. DDS also permits control of the way in which information flows through a space, which is a powerful tool.

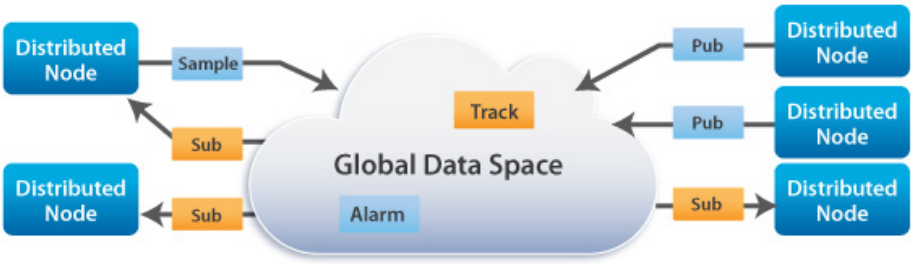

Figure 13. A global data space enables the ability to read and write data asynchronously, anonymously and decoupled in time and space.

DDS is well suited for Industrial Internet and other distributed CPSs in part because of its rich set of QoS policies. QoS policies allow for the control of variables essential to delivering information in a timely and dependable manner. There are about two-dozen QoS policies in DDS that handle priorities, deadlines, data durability, replication and redundancy, history, resource utilization and more(see Fig. 14).

	QoS Policy	QoS Policy	
Volatility	Durability	User Data	User QoS
	History	Topic Data	
	Reader Data Lifecycle	Group Data	
Infrastructure	Writer Data Lifecycle	Partition	Presentation
	Lifespan	Presentation	
	Entity Factory	Destination Order	
	Resource Limits	Ownership	Redundancy
	Reliability	Ownership Strength	
Delivery	Time-Based Filter	Liveliness	
	Deadline	Latency Budget	Transport
	Content Filters	Transport Priority	

Figure 14. DDS includes about two dozen QoS policies.

QoS policies particularly relevant to CPS include the ability to indicate durability, latency, latency bounds and reliability bounds. Likewise, these QoS policies also support the ability to manage availability, coherency issues and resource constraints. There are various actions that can be implemented in this space to gain greater control of CPS.

DDS allows matching of publishers and subscribers in terms of QoS policies that are requested/offered (RxO). This distributed matching capability allows DDS implementations to decide on an optimal way to connect end-to-end flows of producers and consumers. When this capability is integrated on top of an intelligent communication infrastructure, it is able to provide control over the network core.

Since large-scale CPSs do not exist in a vacuum, the ability to bridge different components together is crucial. DDS provides many ways to bridge other technologies through the DDS data bus, which enables communication with web services, Java messaging service and other protocols in a way that can plug and play seamlessly with legacy and new systems. There are also a number of standards available within the DDS ecosystem, such as Java, C++ and UML. It can also take advantage of other standards, including mappings to RESTful web services.

When integrating large-scale CPSs, no single vendor is sufficient. The potential to interwork and connect between parties using a heterogeneous selection of middleware is both valuable and necessary. The OMG Real-Time Publish Subscribe Protocol (RTPS) is the standard Interoperability Wire Protocol used by DDS implementations to enable different vendors to interoperate.

There is currently a vibrant research community focused on DDS. Also, a recently published paper describes various techniques for integrating DDS with WANs, (see http://www.dre.vanderbilt.edu/~schmidt/PDF/DDS-WAN.pdf)

The Big Picture

Despite advances in elastic hardware, it is still hard to deploy CPSs in cloud environments, making the right support necessary. It is improbable that public clouds will be the basis of mission-critical Industrial Internet systems. It is more likely that private clouds will be used, but that does not mean those systems will not benefit from standards and other technologies.

According to the experts, what matters most in computing clouds for CPSs is multi-tenancy and elasticity. Virtualization is beneficial if it can be afforded, but the goal is to run on the bare hardware using powerful integrative middleware technologies, such as DDS. DDS is an ideal fit for distributed CPSs because it is standards-based and includes a number of open-source solutions that facilitate the mixing and matching of capabilities and the ability to build infrastructure for dependable cyber-physical systems.

Although great progress has been made, there remain many research challenges surrounding CPSs. Despite these challenges, DDS is still the most closely matched to the CPS domain and capable of providing off-the-shelf solutions that address these challenges (see http://www.industrialinternet.com/blog/three-qs-professor-douglas-schmidt/).

About Real-Time Innovations

RTI is the real-time distributed infrastructure software company. Our messaging forms the core nervous system for the most demanding real-time applications in the Internet of Things. With RTI Connext™ DDS, devices seamlessly share information and work together as one integrated system.

RTI applications span air, sea, land and space defense; medical imaging, emergency response and hospital integration; power control and energy management; avionics, unmanned systems and air-traffic control; financial and asset trading; and automotive testing and safety.

RTI is committed to open standards, open community source and open architecture. RTI provides the leading implementation of the Object Management Group (OMG) Data Distribution Service (DDS) standard.

RTI is the world's largest embedded middleware provider. RTI is privately held and headquartered in Sunnyvale, California.

About the Author

Dr. Douglas C. Schmidt is a professor of computer science at Vanderbilt University and a widely published author in the fields of object-oriented design and programming, distributed real-time and embedded computing, cyber-physical systems, and software patterns and frameworks. He is co-teaching the first trans-institution massive open online course (MOOC) specialization on mobile cloud computing with Android (see http://www.coursera.org/course/posa).

Your systems. Working as one

CORPORATE HEADQUARTERS
232 E. Java Drive
Sunnyvale, CA 94089

Tel: +1 (408) 990-7400
Fax: +1 (408) 990-7402
info@rti.com

www.rti.com