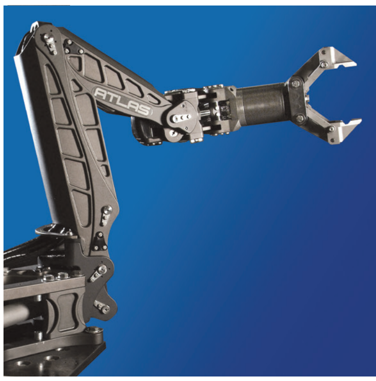
LEADING

APPLICATIONS & ARCHITECTURE

for the

INDUSTRIAL INTERNET OF THINGS



STAN SCHNEIDER, PHD

CEO OF REAL-TIME INNOVATIONS, INC.
SUNNYVALE, CALIFORNIA

rti

LEADING APPLICATIONS AND ARCHITECTURE FOR THE INDUSTRIAL INTERNET OF THINGS (IIOT)

Stan Schneider, PhD. CEO of Real-Time Innovations, Inc. Sunnyvale, California

INTRODUCTION TO THE IIOT

The Internet of Things (IoT) is the name given to the future of connected devices. There are two clear subsets. The "Consumer IoT" includes wearable computers, smart household devices, and networked appliances. The "Industrial IoT (IIoT)" includes networked smart power, manufacturing, medical, and transportation. Technologically, the Consumer IoT and the Industrial IoT are more different than they are similar.

The Consumer IoT attracts more attention because it is more understandable to most people. Consumer systems typically connect only a few points, for instance, a watch or thermostat to the cloud. Reliability is not usually critical. Most systems are "greenfield," meaning there is no existing infrastructure or distributed design that must be considered. There are many exciting new applications that will change daily life. However, the Consumer IoT is mostly a natural evolution of connectivity from human-operated computers to automated things that surround humans.

While it will grow slower than the Consumer IoT, the IIoT will eventually have much larger economic impact. The IIoT will bring entirely new infrastructures to our most critical and impactful societal systems. The opportunity to build truly intelligent distributed machines that can greatly improve function and efficiency across virtually all industries is indisputable. The IIoT is the strategic future of most large companies, even traditional industrial manufacturers and infrastructure providers. The dawn of a new age is clear.

Unlike connecting consumer devices, the IIoT will control expensive, mission-critical systems. Thus, the requirements are very different. Reliability is often a huge challenge. The consequences of a security breach are vastly more profound for the power grid than for a home thermostat. Existing industrial systems are already networked in some fashion, and interfacing with these legacy "brownfield" designs is a key blocking factor. Plus, unlike consumer devices that are mostly connected on small networks, industrial plants, electrical systems or transportation grids will encompass many thousands or millions of interconnected points.

Building a technology stack for any one of these applications is a challenge. However, the real power is a single architecture that can span sensor-to-cloud, interoperate between vendors and span industries. The challenge is to evolve from today's mashup of special-purpose standards and technologies to a fast, secure, interoperable future.

In the long term, there is an even larger opportunity. The future of the IIoT will include enterprise-class platforms that guarantee real-time delivery across enterprises and metro or continental areas. This will become a new utility that enables reliable distributed systems. This utility will support twenty-first-century infrastructure like intelligent transportation with autonomous vehicles and traffic control, smart grids that integrate distributed energy resources, smart healthcare systems that assist care teams and safe flying robot air traffic control systems. This utility will be as profound as the cell phone network, GPS or the Internet itself.

There are many consortia of companies targeting the IIoT. The largest and fastest growing is the Industrial Internet Consortium (IIC)[1]. The IIC was founded in 2014 by global industrial leaders: GE, Intel, Cisco, AT&T and IBM. As of this writing in 2016, it includes over 250 members. The German government, along with several large German manufacturers, has an active effort called Industrie 4.0.[2] There is also a smaller startup consortium called the OpenFog Consortium.[3] Of these, the IIC is by far the broadest. It addresses end-to-end designs in all industries. Industrie 4.0 is focused only on manufacturing. And OpenFog targets "intelligence at the edge," meaning the movement of powerful, elastic computing out of data centers into neighborhoods and customer premises. However, all share many common members and goals. They are working together in many ways.

1 http://iiconsortium.org

2 https://en.wikipedia.org/wiki/Industry_4.0

3 www.openfogconsortium.org

Because of its size and growth, the IIC gets by far the most attention. The goal of the IIC is to develop and test an architecture that will span all industries. Just as Ethernet, Linux and the Internet itself grew as general-purpose technologies that pushed out their special-purpose predecessors, the IIC will build a general-purpose Industrial Internet architecture that can build and connect systems such as transportation, medical, power, factory, industrial controls and others. The IIC's unique and powerful combination of leaders from both government and industry gives it the necessary platform to make this huge impact.

The IIC was the first to create a venue for users across industries with similar challenges. This actually created the IIoT as a true market category and changed the landscape dramatically. Suddenly, hundreds of companies are deciding their strategy for this new direction. Gartner, the large analyst firm, predicts that the Smart Machine era will be the most disruptive in the history of IT. That disruption will be led by smart distributed infrastructure called the IIoT.

SOME EXAMPLES IIOT APPLICATIONS

The author is the CEO of Real-Time Innovations, Inc. (RTI).[4] RTI is the largest vendor of embedded middleware company and the leading vendor of middleware compliant with the Data Distribution Service (DDS) standard.[5]

All applications in this section are operational RTI Connext DDS systems. The DDS standard is detailed in later sections; the applications are presented first to provide background and highlight the breadth of the IIoT challenge. These are only a few of nearly 1,000 applications. Further examples can be found at **www.rti.com**.

4 http://www.rti.com

5 http:/www.omg.org/dds

Connected Medical Devices for Patient Safety

Thirty years ago, health care technologists realized a simple truth: monitoring patients improves outcomes. That epiphany spawned the dozens of devices that populate today's hospital rooms: pulse oximeters, multi-parameter monitors, ECG monitors, Holter monitors, and more. Over the ensuing years, technology and intelligent algorithms improved many other medical devices, from infusion pumps (IV drug delivery) to ventilators. Healthcare is much better today because of these advances.

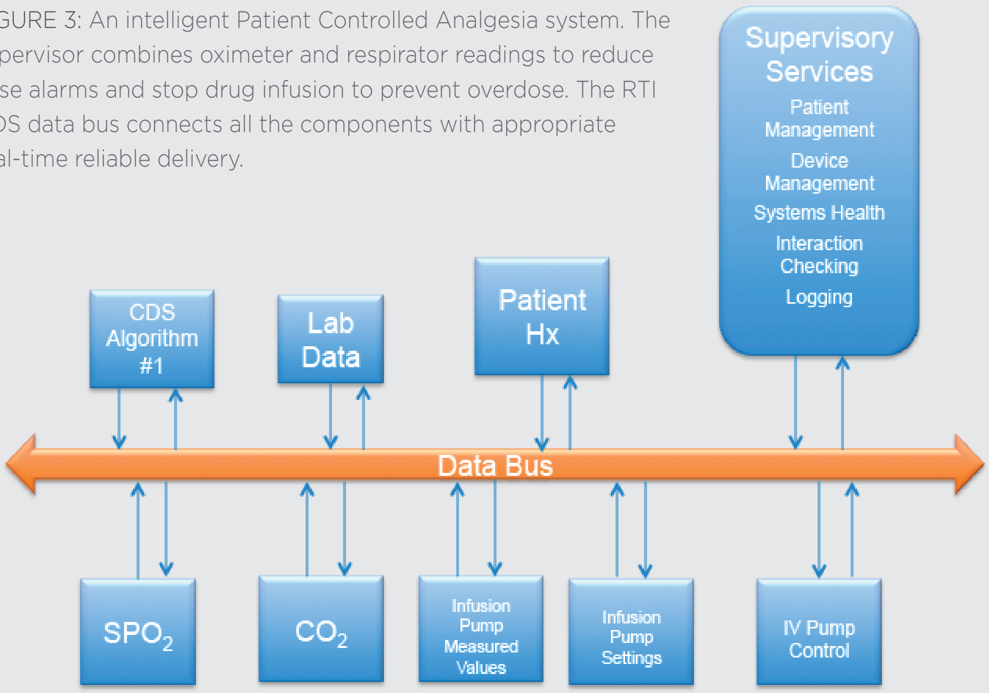
However, hospital error is still a leading cause of death; in fact, the Institute of Medicine named it the third leading cause of death after heart disease and cancer. Thousands and thousands of errors occur in hospitals every day. Many of these errors are caused by false alarms, slow responses, and inaccurate treatment delivery.

Today, a new technology disruption is spreading through patient care: intelligent, distributed medical systems. By networking devices, alarms can become smart, only sounding when multiple devices indicate errant physiological parameters. By connecting measurements to treatment, smart drug delivery systems can react to patient conditions much faster and more reliably than busy hospital staff. By tracking patients around the hospital and connecting them to cloud resources, efficiency of care can be dramatically improved. The advent of true Internet of Things networking in healthcare will save costs and lives.

FIGURE 1: Connected medical devices will intelligently analyze patient status, create "smart alarms" by combing instrument readings, and ensure proper patient care. An intelligent, distributed IIoT system will help care teams prevent hundreds of thousands

FIGURE 2: A modern hospital needs hundreds of types of devices. These must communicate to improve patient safety and outcome, to aid resource deployment and maintenance, and to optimize business processes. RTI Connext DDS adapts to handle many different types of dataflows, different computing platforms and transports.

FIGURE 3: An intelligent Patient Controlled Analgesia system. The supervisor combines oximeter and respirator readings to reduce false alarms and stop drug infusion to prevent overdose. The RTI DDS data bus connects all the components with appropriate real-time reliable delivery.

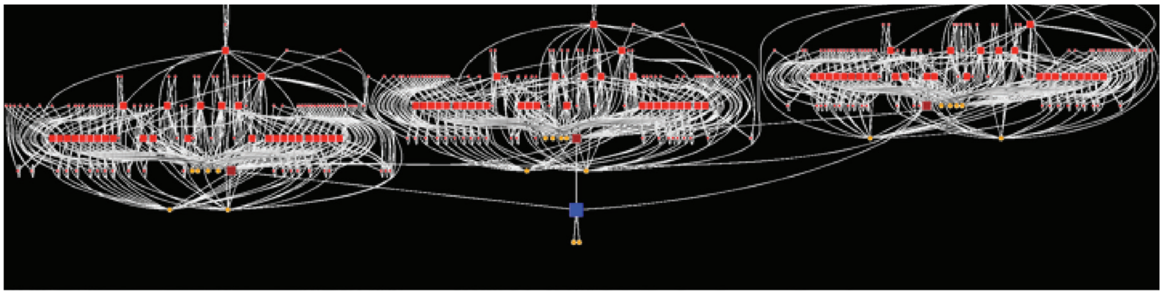


THE INTEGRATED CLINICAL ENVIRONMENT (ICE)

Researchers and device developers are making quick progress on medical device connectivity. The Integrated Clinical Environment standard (ASTM F2761)[6] is one key effort to build such a connected system. ICE combines standards. It takes data definitions and nomenclature from the IEEE 11073 (x73) standard for health informatics. It specifies communication via the Data Distribution Service (DDS) standard. ICE then defines via the DDS standard control, data logging and supervisory functionality to create a connected intelligent substrate for smart clinical connected systems.

Like most standards, large organizations may take time to adapt to a standards-driven environment. ICE nonetheless represents an excellent example of how to build smarter systems.

6 http://www.icealliance.org/



PATIENT MONITORING

Modern hospitals use hundreds of types of devices for patient care and monitoring. These systems must work in a large hospital environment. Integrating whole hospitals with thousands of devices presents challenges for scalability, performance and data discovery.

To prove the design viable for GE Healthcare, RTI built a simulation to prove that DDS could handle a thousand-bed hospital with over 100,000 devices. The simulation ran in RTI's networking lab. It sent realistic dataflows between hundreds of applications, instances of RTI's Connector product. RTI services developed a matrix (Excel spreadsheet) to configure Connector to send the mix of data types and rates expected from real devices. RTI developed an automated test harness to deploy these applications across the lab's test computers and collect the results. A graph of part of the simulation topology is presented in the following text. RTI's test harness collected dataflow rates and loading across this topology.

The system handled realistic scale, performance and discovery. Since it is important to communicate real-time waveforms and video, the potential network-wide dataflow is large. However, the need is "sparse;" most data is only needed at relatively few points. As explained in the succeeding text, DDS can propagate specifications to the senders to indicate exactly what each receiver needs from the senders. The senders then filter the information to send only what's needed, thereby eliminating wasted bandwidth. Discovering data sources is also critical, since 62% of hospital patients move every day. So, the system also tested transitions between network locations.

FIGURE 4: Medical devices must operate in a complex hospital environment. The system must be able to find data sources, track them as patients move, and scale to handle the load. This realistic test simulated a large hospital.

When deployed, the new system will ease patient tracking. It will coordinate devices in each room and connect rooms into an integrated whole hospital. Information will flow easily and securely to cloud-based Electronic Health Records (EHR) databases. The hospital of the future will become an intelligent, distributed machine in the IIoT.

Microgrid Power Systems

The North American electric power grid has been described as the biggest machine in the world. It was designed and incrementally deployed based on centralized power generation, transmission and distribution concepts.

Times have changed. Instead of large, centralized power plants burning fossil fuels that drive spinning masses, Distributed Energy Resources (DERs) have emerged as decentralized, local alternatives to bulk power. DERs are typically clean energy solutions (solar, wind, thermal) that take advantage of local environmental and market conditions to manage the local generation, storage or consumption of electricity.

THE DER TIME CHALLENGE

Most renewable energy sources are not reliable producers. Solar and wind can change their power output very quickly.

Unfortunately, that dynamic behavior is not compatible with today's grid. Today's grid uses local power substations to convert high-voltage power to neighborhood distribution voltage levels. Those stations estimate power needs and reports back to the utility. The utility then needs up to 15 minutes to spin up (or down) a centralized generation plant to match the estimate.

So, since a solar array can lose power in a matter of seconds with a fast moving cloud, the grid cannot react. An alternate source has to be available and ready to pick up the load immediately. If there isn't sufficient backup, the voltage on the grid can drop and the grid can fail. The only way to provide that backup today is to provide "spinning reserve" capacity, meaning the generators use more energy than the grid needs.

As solar energy resources grow in a utility's service area, the utility has to have more excess spinning reserve ready as backup. While the sun is shining, power may be flowing from these distributed solar arrays back to the grid. However, the fossil fuel generators need to be running and spun up sufficiently to quickly take over if the solar arrays stop producing. So, with every solar array pushing power on to the grid, there is an equivalent fossil fuel generator spinning in the background to take over. Thus, little fossil fuel is saved. Even worse, driving the generators without load makes them overheat and even prematurely wears out bearings.

To fix this, the utility needs 15-30 minutes of extra time to ramp up the generators. Then, they would not need to have the spinning reserve. The only way to provide the time needed is to implement energy storage or load reduction.

MICROGRID ARCHITECTURE

Microgrids are the leading way to provide that time. Microgrids combine intermittent energy sources, energy storage systems like batteries, and some local control capability. This allows the microgrid to smooth out the changes in DER power. A microgrid can even "island" itself from the main power grid and run autonomously.

Microgrids usually encompass a well-defined, relatively small geographic region. College campuses have been proving grounds for this technology, as have military bases. A microgrid can respond rapidly and locally to a loss of power from solar arrays or local wind turbines using backup energy sources like batteries. Many proof-of-concept microgrid projects are active; they range from small

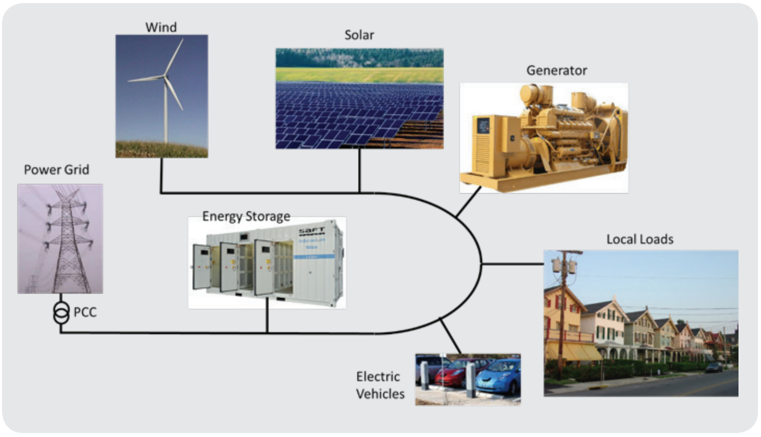


FIGURE 5: A Microgrid uses peer-to-peer data communication and edge intelligence to automate local power generation and balance against the power load. Microgrids help integrate intermittent energy sources like solar and wind.

demos to utility-class pilot testbeds. All seek to incorporate energy storage and load reduction techniques into the grid.

Two key capabilities for microgrids are intelligent control at the edge of the grid and peer-to-peer, high-performance communications for local autonomy (see Figure 5). With these, a local battery energy storage system can receive a message in milliseconds from the solar arrays when backup energy is needed. The local controller on the battery can then quickly switch the battery from charge to source mode. This keeps the local energy consumers powered and gives the utility time to spin up central power resources as needed.

The OpenFMB™ Framework[7] is the first field system addressing the need for reliable, safe, upgradeable distributed intelligence on the grid. OpenFMB directly addresses the decentralization issue facing utilities and regulators by leveraging existing electricity information models (e.g., IEC 61968/61970, IEC 61850, MultiSpeak and SEP 2) and creating a data-centric "bus" on the grid to allow devices to talk directly to one another. The initial use cases targeted by OpenFMB are microgrid focused, and the OpenFMB framework is closely adhering to the IIC's Industrial Internet Reference Architecture IIRA.

The OpenFMB team held a major demonstration in February 2016 with 25 different companies. Many parts of

7 http://sgip.org/Open-Field-Message-Bus-OpenFMB-Project

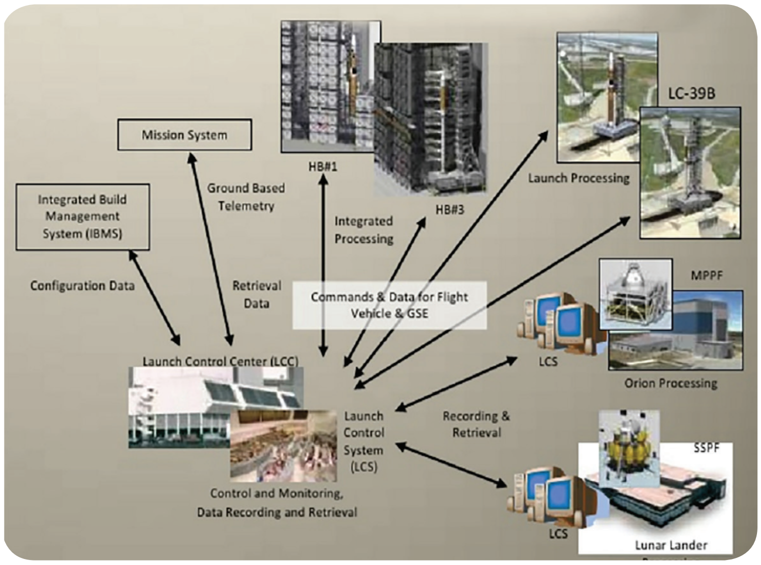

Mission System
Integrated Build Management System (IBMS)
Ground Based Telemetry
HB#1
HB#3
Integrated Processing
Launch Processing
LC-39B
Configuration Data
Retrieval Data
Commands & Data for Flight Vehicle & GSE
MPPF
LCS
Orion Processing
Launch Control Center (LCC)
Launch Control System (LCS)
Recording & Retrieval
SSPF
Control and Monitoring, Data Recording and Retrieval
LCS
Lunar Lander

FIGURE 6: NASA KSC's launch control is a massive, reliable SCADA system. It comprises over 400,000 points, spread across the launch platform and the control room. The launch control system integrates many thousands of devices, from tiny sensors to large enterprise storage systems. It spreads over many miles.

Photo: NASA/Bill Ingalls

the implementation use RTI's Connext DDS platform. DDS interfaces were developed for the Optimization Engine, Load Simulators, the Point of Common Coupling (PCC) transition logic, and other required simulators to drive the demonstration. To test non-proprietary interoperability, the system built a cross-platform solution with multiple operating system targets and CPU architectures. The demonstration proves that IIoT interoperability is a practical, achievable path for fielded utility devices and systems.

Large-Scale SCADA Control

NASA Kennedy Space Center's launch control system is the largest SCADA (Supervisory Control And Data Acquisition) system in the world. With over 400,000 control points, it connects together all the equipment needed to monitor and prep the rocket systems. Before launch, it pumps rocket fuels and gasses, charges all electrical systems, and runs extensive tests. During launch, a very tightly controlled sequence enables the main rocket engines, charges and arms all the attitude thrusters, and monitors thousands of different values that make up a modern space system. It must also adapt to the various mission payloads, some of which need special preparation and monitoring for launch.

The launch control system has very tight and unique communications requirements. The system is distributed over a large area and the control room. It must be secure. Dataflow is "tidal:" activity cycles through the surge of preparation, spikes during the actual launch, then ebbs afterward. During the most critical few seconds, it sends hundreds of thousands of messages per second. Connext DDS intelligently batches updates from thousands of sensors, reducing traffic dramatically. Everything must be stored for later analysis. All information is viewable (after downsampling) on HMI stations in the control room. After launch, all the data must be available for replay, both to analyze the launch and to debug future modifications in simulation.

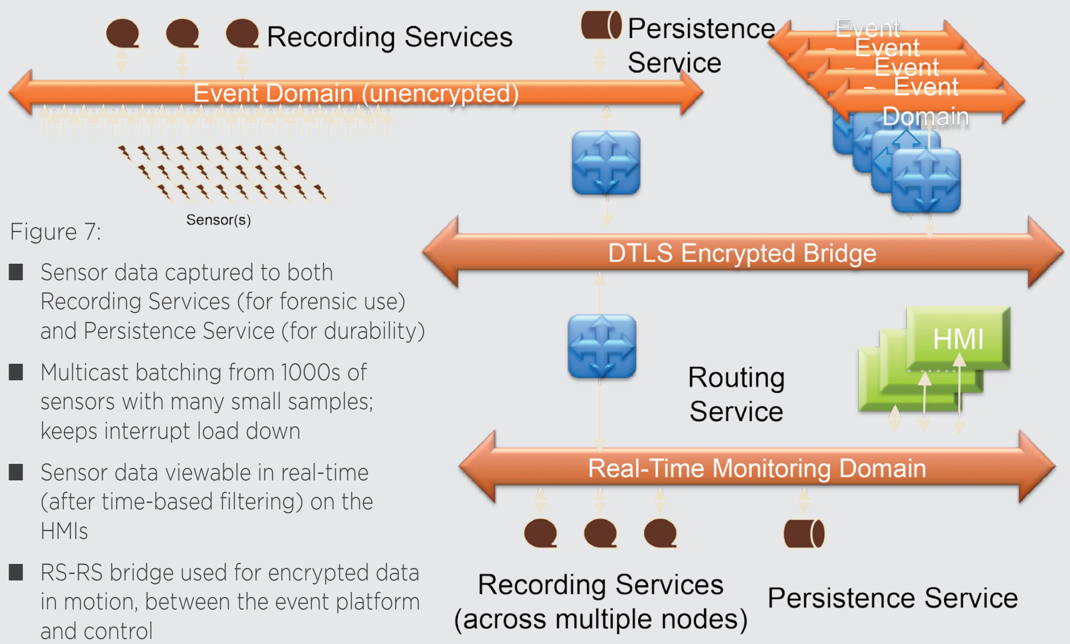


Figure 7:

- Sensor data captured to both Recording Services (for forensic use) and Persistence Service (for durability)
- Multicast batching from 1000s of sensors with many small samples; keeps interrupt load down
- Sensor data viewable in real-time (after time-based filtering) on the HMIs
- RS-RS bridge used for encrypted data in motion, between the event platform and control

Autonomy

RTI was founded by researchers at the Stanford Aerospace Robotics Laboratory (ARL)[8]. The ARL studies complex electromechanical systems, especially those with increasing levels of autonomy.

Unmanned air and defense vehicles have long relied on DDS for deployments on land, in the air, and underwater. DDS is a key technology in many open architecture initiatives, including the Future Airborne Capability Environment (avionics), Unnamed Air Systems (UAS) Control Segment Architecture, UAS ground stations) and the Generic Vehicle Architecture (military ground vehicles).

UAS have complex communication requirements, with flight-critical components distributed across the air and ground segments. Further, to operate in the U.S. National Airspace System (NAS) the system must be certified to the same safety standards as civil aircraft. RTI recently announced a version of DDS with full DO-178C Level A safety certification evidence. It was developed to meet the needs of the Ground Based Sense and Avoid (GBSAA) system pictured in Figure 8.

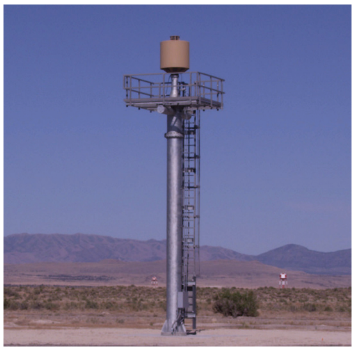
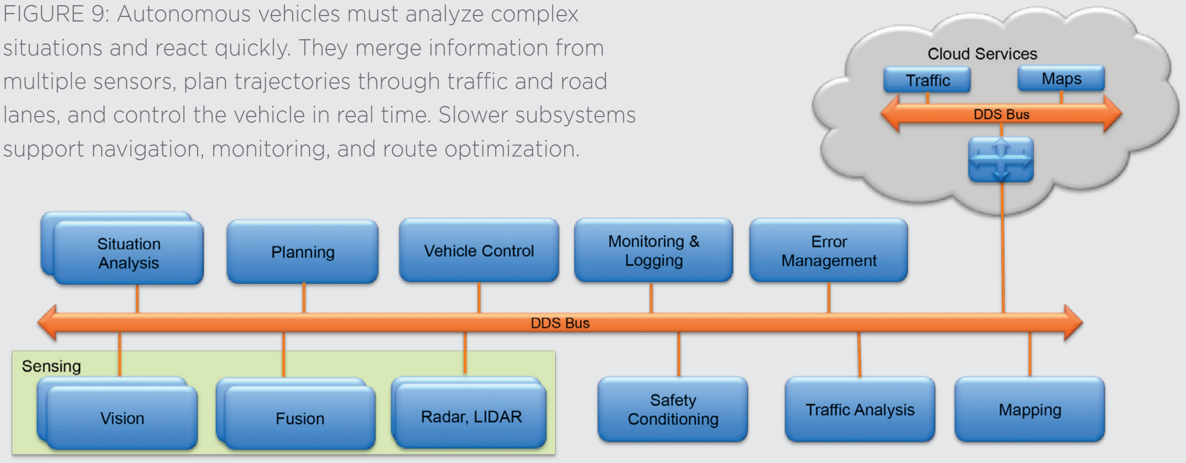


Figure 8: The Ground Based Sense and Avoid (GBSAA) system includes many distributed radars. It will soon allow unmanned vehicles to fly in the U.S. National Airspace System (NAS). Applications of unmanned vehicles will include operator training, repositioning, search and rescue, and disaster relief.

US Army photo by Sofia Bledsoe.

8 http://sgip.org/Open-Field-Message-Bus-OpenFMB-Project

FIGURE 9: Autonomous vehicles must analyze complex situations and react quickly. They merge information from multiple sensors, plan trajectories through traffic and road lanes, and control the vehicle in real time. Slower subsystems support navigation, monitoring, and route optimization.



This technology is now also being applied aggressively to autonomous cars for consumer use. This market is perhaps the most disruptive of the IIoT applications. Because the ground is a much more complex environment, autonomous cars face even greater challenges than air systems. They must coordinate navigation, traffic analysis, collision detection and avoidance, high-definition mapping, lane departure tracking, image and sensor processing, and more.

Safety certification for the entire system as whole is prohibitively expensive. Dividing the system into modules and certifying them independently reduces the cost dramatically. "Separation kernels" are operating systems that provide guaranteed separation of tasks running on one processor. "Separation middleware" provides a similar function to applications that must communicate, whether they are running on one processor or in a distributed system. A clean, well-controlled interface eases certification by enabling modules to work together.

TOWARD A TAXONOMY OF THE IIOT

There is today no organized system science for the IIoT. We have no clear way to classify systems, evaluate architectural alternatives, or select core technologies. To address the space more systematically, we need to develop a taxonomy of IIoT applications based on their system requirements.

This taxonomy will reduce the space of requirements to a manageable set by focusing only on those that drive significant architectural decisions. Based on extensive experience with real applications, we suggest a few divisions and explain why they impact the architecture. Each of these divisions defines an important dimension of the IIoT taxonomic model. We thus envision the IIoT space as a multi-dimensional requirement space. This space provides a framework for analyzing the fit of architectures and technologies to IIoT applications.

A taxonomy logically divides types of systems by their characteristics. The first problem is to choose top-level divisions. In the animal kingdom, you could label most animals "land, sea or air" animals. However, those environmental descriptions don't help much in understanding the animal. For instance, the "architecture" of a whale is not much like an octopus, but it is very like a bear. To be understood, animals must be divided by their characteristics and architecture, such as heart type, reproductive strategies, and skeletal structure.

It is similarly not useful to divide IIoT applications by their industries like "medical, transportation and power." While these environments are important, the requirements simply do not split along industry lines. For instance, each of these industries has some applications that must process huge data sets, some that require real-time response, and others that need life-critical reliability. Conversely, systems with vastly different requirements exist in each industry. The bottom line is that fundamental system requirements vary by application and not by industry, and these different types of systems need very different approaches.

FIGURE 10: Environment does not indicate architecture. Dividing animals by "land, sea, and air" environment is scientifically meaningless. The biological taxonomy instead divides by fundamental characteristics.

Thus, as in biology, the IIoT needs an environment-independent system science. This science starts by understanding the key system challenges and resulting requirements. If we can identify common cross-industry requirements, we can then logically specify common cross-industry architectures that meet those requirements. That architecture will lead to technologies and standards that can span industries.

There is both immense power and challenge in this statement. Technologies that span industries face many challenges, both political and practical. Nonetheless, a clear fact of systems in the field is the similarity of requirements and architecture across industries. Leveraging this fact promises a much better understood, better connected future. It also has immense economic benefit: over time, generic technologies offer huge advantage over special-purpose approaches. Thus, to grow our understanding and realize the promise of the IIoT, we must abandon our old industry-specific thinking.

Proposed Taxonomic Criteria

So, what can we use for divisions? What defining characteristics can we use to separate the mammals from the reptiles from the insects of the IIoT?

There are far too many requirements, both functional and non-functional, to consider in developing a "comprehensive" set to use as criteria. As with animals, we need to find those few requirements that divide the space into useful, major categories.

The task is simplified by the realization that the goal is to divide the space so we can determine system architecture. Thus, good division criteria are (1) unambiguous and (2) impactful on the architecture. That makes the task easier, but still non-trivial. The only way to do it is through experience. We are early in our quest. However, significant progress is within our collective grasp.

This work draws on extensive experience with nearly 1,000 real-world IIoT applications. Our conclusion is that an IIoT taxonomy is not only possible but also critical to both the individual systems building and the inception of a true cross-industry IIoT.

While the classification of IIoT systems is very early, we do suggest a few divisions. To be as crisp as possible, we

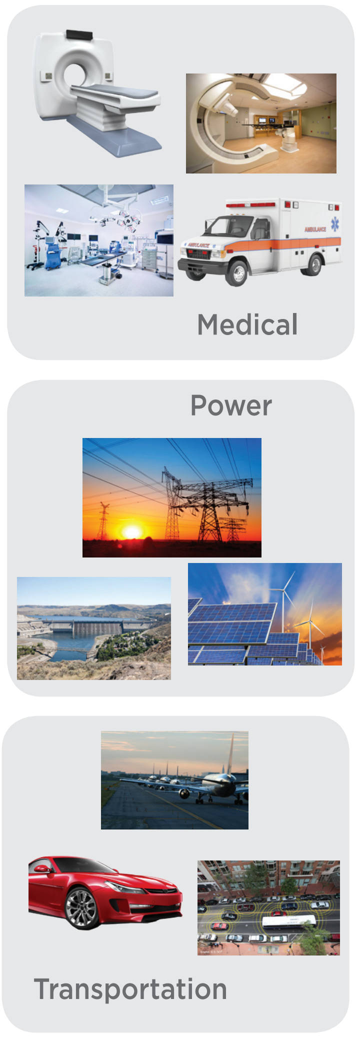


FIGURE 11: Industry does not indicate architecture. Dividing IIoT applications by "medical, power, or transportation" environment is as scientifically meaningless as dividing animals by their environments. To make progress, we need an IIoT taxonomy that instead divides by fundamental characteristics.

also chose numeric "metrics" for each division. The lines, of course, are not that stark. And those lines evolve with technology over time at a much faster pace than biological evolution. Nonetheless, the numbers are critical to force clarity; without numerical metrics, meaning is often too fuzzy.

RELIABILITY

Metric: Continuous availability must exceed "99.999%" to avoid severe consequences

Architectural Impact: Redundancy

Many systems describe their requirements as "highly reliable," "mission critical" or "minimal downtime." However, those labels are more often platitudes than actionable system requirements. For these requirements to be meaningful, we must be more specific about the reasons we must achieve that reliability. That requires understanding how quickly a failure causes problems and how bad those problems are.

Thus, we define "continuous availability" as the probability of a temporary interruption in service over a defined system-relevant time period. The "five 9s" golden specification for enterprise-class servers translates to about 5 minutes of downtime per year. Of course, many industrial systems cannot tolerate even a few milliseconds of unexpected downtime. For a power system, the relevant time period could span years. For a medical imaging machine, it could be only a few seconds.

The consequences of violating the requirement are also meaningful. A traffic control system that goes down for a few seconds could result in fatalities. A website that goes down for those same few seconds would only frustrate users. These are fundamentally different requirements.

Reliability thus defined is an important characteristic because it greatly impacts the system architecture. A system that cannot fail, even for a short time, must support redundant computing, sensors, networking, storage, software and more. Servers become troublesome single-point-of-failure weak points. When reliability is truly critical, redundancy quickly becomes a—or perhaps *the*—key architectural driver.

FIGURE 12: IIoT reliability-critical applications. Hydropower dams can quickly modulate their significant power output by changing water flow rates and thus help balance the grid; even a few milliseconds of unplanned downtime can threaten stability. Air-traffic control faces a similar need for continuous operation; a short failure in the system endangers hundreds of flights. A proton-beam radiation therapy system must guarantee precise operation during treatment; operational dropouts threaten patient outcomes. Applications with severe consequences of short interruptions in service require a fully-redundant architecture, including computing, sensors, networking, storage and software.

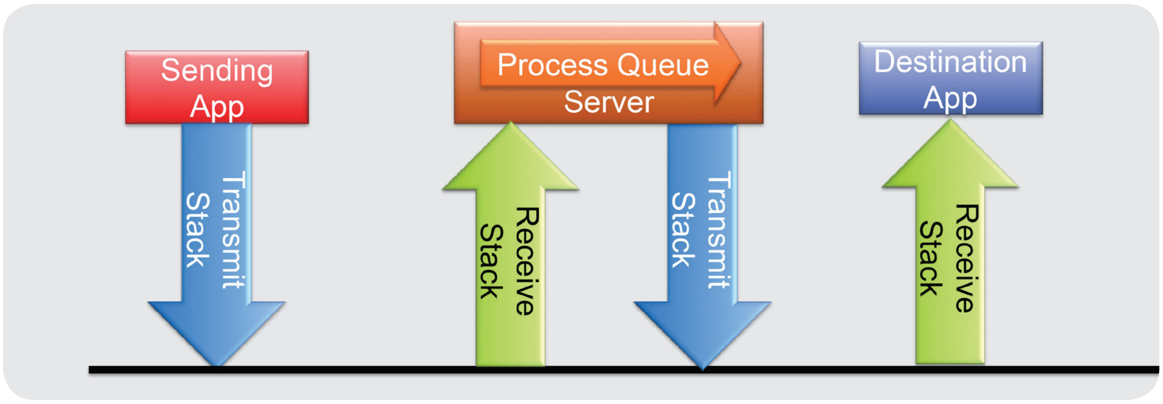


FIGURE 13: Added Server Latency. Although the hardware transmit time is often negligible, sending data through a server "hop" requires traversing the sending machine's transmit stack, the server's receive stack, the server's processing queue, the server's transmit stack and finally the destination's receive stack. Each of these has threads, queues and buffers that add uncontrolled latency. Worse, the server cannot easily prioritize traffic as easily as the end points. Thus, systems that are sensitive to maximum latency often cannot use data servers.

REAL TIME

Metric: Response < 100ms

Architectural Impact: Peer-to-peer data path

There are many of ways to characterize "real time." All systems should be "fast." However, for these requirements to be useful we must specifically understand which timing requirements drive success.

Thus, "real time" is much more about guaranteed response than it is about fast. Many systems require low average latency (delivery delay). However, true real-time systems succeed only if they always respond "on time." This is the maximum latency, often expressed as the average delay plus the variation or "jitter." Even a fast server with low average latency can experience large jitter under load.

In a distributed system, the most important architectural impact is the potential jitter imposed by a server or broker in the data path. An architecture that can satisfy a human user annoyed by a wait longer than eight seconds for a website will never satisfy an industrial control that must respond in 2 milliseconds. We find that the "knee in the curve" that greatly impacts design occurs when the speed of response is measured in a few tens of milliseconds or even microseconds. We choose 100 ms, simply because that is about the unpredictable delay of today's servers. Systems that most respond faster than this usually must be peer-to-peer, and that is a huge architectural impact.

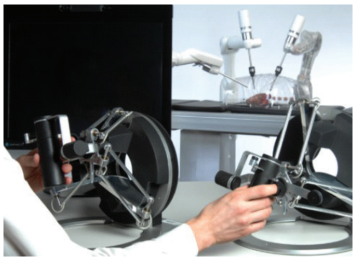
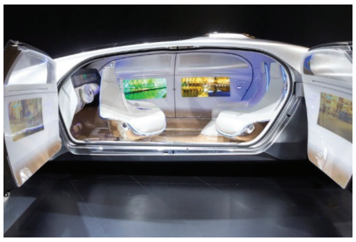



FIGURE 14: IIoT real-time applications. To provide quality feel to surgeons, distributed control loops for medical robotics must run at rates up to 3 kHz and control the "jitter" to only tens of microseconds. Similarly, autonomous cars must react fast enough to safely control the vehicle and prevent collisions. These fundamental performance needs imply a system architecture that does not send data through intermediaries.

Robotics photo: DLR CC-BY 3.

DATA ITEM SCALE

Metric: More than 10,000 addressable data items

Architectural Impact: Selective delivery filtering

Scale is a fundamental challenge for the IIoT. It is also complex; there are many dimensions of scale, including number of "nodes," number of applications, number of developers on the project, number of data items, data item size, total data volume, and more. We cannot divide the space by all these parameters.

In practice, however, they are related. For instance, a system with many data items probably has many nodes.

Despite the broad space, we have found that two simple metrics correlate well with architectural requirements.

The first scale metric is addressable "data item scale," defined as the number of different data instances that could be of interest to different parts of the system. Note that this is not the same as the size of a single large data set, such as a stream of data from a single fast sensor. The key scale parameter is the existence of many different data items that could potentially be of interest to different consumers. So, a few fast sensors create only a few addressable data items. Many sensors or sources create many data items. A large number of addressable data items implies difficulty in sending the right data to the right place.

When systems get "big" in this way, it is no longer practical to send every data update to every possible receiver. We find that the challenge is significant for as few as 100 data items. It is extreme for systems with more than 10,000 addressable data items. Above this limit, managing the data itself becomes a key architectural need. These systems need an architectural design that explicitly understands the data, thereby allowing selective filtering and delivery. There are two approaches in common use: run-time introspection that allows consumers to choose data items themselves, and "data-centric" designs that empower the infrastructure itself to understand and actively filter the data system-wide.

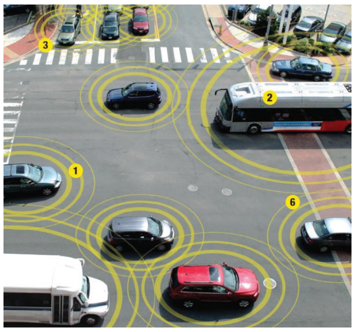


Figure 15: IIoT applications with many data items. IIoT systems often produce far too much data to send everything to every possible consumer. "Gust control" in a wind turbine farm, for instance, needs weather updates from the turbines immediately "up wind," a specification that changes with time. Traffic control systems are very interested only in vehicles approaching an intersection. These applications require the architecture to provide selective data availability, so only the right information loads the network and the participants.

MODULE SCALE

Metric: More than 10 teams or interacting applications

Architectural Impact: Interface control and evolution

The second scale parameter we choose is the number of "modules" in the system, where a module is defined as a reasonably independent piece of software. Each module is typically an independently-developed application built by an independent team of developers on the "project."

Module scale quickly becomes a key architectural driver. The reason is that system integration is inherently an "n-squared" problem. Each new team presents another interface into the system. Smaller projects built by a cohesive team can easily share interface specifications without formality. Larger projects built by many independent groups of developers face a daunting challenge. System integration can occupy half of the delivery schedule and most of its risk.

In these large systems, interface control dominates the interoperability challenge. It is not practical to expect interfaces to be static. Modules, or groups of modules, that depend on an evolving interface schema must somehow continue to interoperate with older versions of that schema. Communicating all the interfaces becomes hard. Forcing all modules to "update" on a coordinated timeframe to a new schema becomes impossible. Thus, interacting teams quickly find that they need tool, process, and eventually architectural support to solve the system integration problem.

Of course, this is a well-studied problem in enterprise software systems. In the storage world, databases ease system integration by explicitly modeling and controlling "data tables," thus allowing multiple applications to access information in a controlled manner. Communication technologies like enterprise service buses (ESBs), Web services, enterprise "queuing" middleware, and textual schema like XML and JSON all provide evolvable interface flexibility. However, these are often not appropriate for industrial systems, usually for performance or resource reasons.

Data-centric systems expose and control interfaces directly, thus easing system integration. Databases, for instance, provide data-centric storage and are thus important in systems with many modules. However, databases provide

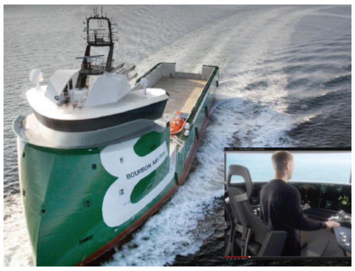


FIGURE 16: IIoT applications built by large teams. Hundreds of different types of hospital medical devices, from heart monitors to ventilators, must combine to better monitor and care for patients. Similarly, ship systems integrate dozens of complex functions like navigation, power control and communications. When a complex "system of systems" integrates many complex interfaces, the system architecture itself must help to manage system integration and evolution.

storage for data at rest. Most IIoT systems require data in motion, not (or in addition to) data at rest.

Data-centric middleware is a relatively new concept for distributed systems. Similar to a database data table, data-centric middleware allows applications to interact through explicit data models. Advanced technologies can even detect and manage differences in interfaces between modules and then adapt to deliver to each endpoint in the schema what the endpoint expects.[9] These systems thus decouple application interface dependencies, allowing large projects to evolve interfaces and make parallel progress on multiple fronts.

RUNTIME INTEGRATION

Metric: More than 20 "devices," each with many parameters and data sources or sinks that cannot be configured at development time

Architectural Impact: Must provide a discoverable integration model

Some IIoT systems can (or even must) be configured and understood before runtime. This does not mean that every data source and sink is known, but rather that this configuration is relatively static. Others, despite a potentially large size, have applications that implement specific functions that depend on knowing what data will be available. These systems can or must implement an "end point" discovery model that finds all the data in the system directly.

However, other systems cannot easily know what devices or data will be available until runtime. For instance, when IIoT systems integrate racks of field-replaceable machines or devices, they must often be configured and understood during operation. For instance, a plant controller HMI may need to discover the device characteristics of an installed device or rack so a user can choose data to monitor.

The key factor here is not addition or changes in which device is used. It is more a function of not knowing which types of devices may be involved.

FIGURE 17: IIoT device integration challenge. Large systems assembled in the field from a large variety of "devices" face a challenge in understanding and discovering interacting devices and their relationships. The most common example applications are in manufacturing. These applications benefit from a design that offers the ability for remote applications and human interfaces to "browse" the system, thus discovering data sources and relationships.

9 http://www.omg.org/spec/DDS-XTypes/

These systems must implement a different way to discover information. Instead of searching for data, it is more efficient to automate the process by building runtime maps of devices and their data relationships. The choice of "20" different devices is arbitrary. The point is that when there are many different configurations for many devices, mapping them at runtime becomes an important architectural need. Each device requires some sort of server or manager that locally configures attached sub-devices, and then presents that catalog to the rest of the system. This avoids manual gymnastics.

DISTRIBUTION FOCUS

Metric: Fan out > 10

Architectural Impact: Must use one-to-many connection technology

We define "fan-out" as the number of data recipients that must be informed upon change of a single data item. Thus, a data item that must go to 10 different destinations each time it changes has a fan out of "10."

Fan-out impacts architecture because many protocols work through single 1 : 1 connections. Most of the enterprise world works this way, often with TCP, a 1 : 1 session protocol. Examples include connecting a browser to a Web server, a phone app to a backend, or a bank to a credit card company. While these systems can achieve significant scale, they must manage a separate connection to each endpoint. When many data updates must go to many endpoints, the system is not only managing many connections, but it is also sending the same data over and over through each of those connections.

IIoT systems often need to distribute information to many more destinations than enterprise systems. They also often need higher performance on slower machines. Complex systems even face a "fan-out mesh" problem, where many producers of information must send it to many recipients. When fan-out exceeds 10 or so, it becomes impractical to do this branching by managing a set of 1 : 1 connections. An architecture that supports efficient multiple updates greatly simplifies these systems.

FIGURE 18: "IIoT applications needing data distribution. Many applications must deliver the same data to many potential endpoints. Coordinated vehicle fleets may update a cloud server, but then that information must be delivered to many distributed vehicles. An emergency services communications system must allow many remote users access to high-bandwidth distributed voice and video streams. Many industries use "hardware in the loop" simulation to test and verify modules during development. Across all these industries, an efficient architecture must deliver data to multiple points easily.

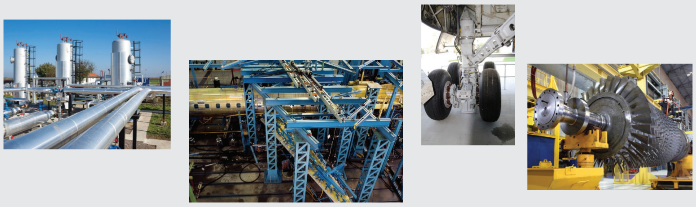


FIGURE 19: IIoT Collection and Monitoring Applications. Collecting and analyzing field-produced data is perhaps the first "killer app" of the IIoT. The IIC's "track and trace" testbed, for instance, tracks tools on a factory floor so the system can automatically log use. Other applications include monitoring gas turbines for efficient operation, testing aircraft landing gear for potentially risky situations and optimizing gas pipeline flow control. Since there is little inter-device flow, "hub and spoke" system architectures that ease collection work well for these systems.

COLLECTION FOCUS

Metric: One-way data flow from more than 100 sources

Architectural Impact: Local concentrator or gateway design

Data collection from field systems is a key driver of the IIoT. Many systems transmit copious information to be stored or analyzed in higher-level servers or the cloud. Systems that are essentially restricted to the collection problem do not share significant data between devices. These systems must efficiently move information to a common destination, but not between devices in the field.

This has huge architectural impact. Collection systems can often benefit from a hub-and-spoke "concentrator" or gateway. Widely distributed systems can use a cloud-based server design, thus moving the concentrator to the cloud.

Dimensional Decomposition and Map to Implementation

The analogy with a biological taxonomy only goes so far. Industrial systems do not stem from common ancestors and thus do not fall into crisply-defined categories. As implied previously, most systems exhibit some degree of each of the characteristics. This is actually a source of much of the confusion, and the reason for our attempt to choose hard metrics at the risk of declaring arbitrary boundaries. In the end, however, the goal is to use the characteristics to help select a single system architecture. Designs and technologies satisfy the previously mentioned goals to varying degrees. With no system science to frame the search, the selection of a single architecture based on any one requirement becomes confusing.

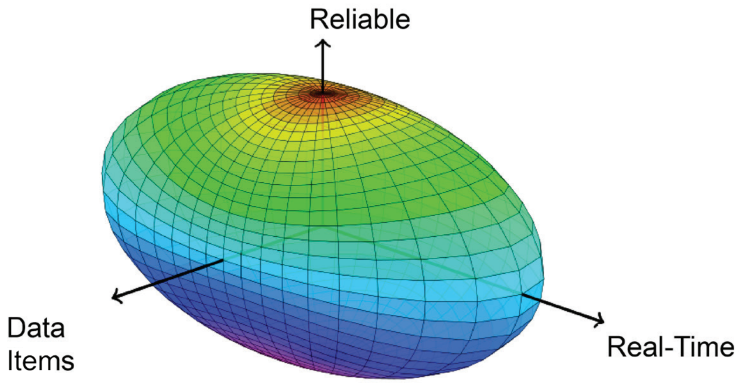


FIGURE 20: n-Dimensional Requirement Space. Architectural approaches and their implementing technologies satisfy some range of each of the dimensions above, and thus occupy a region in an *n*-dimensional "requirement space." The value of a taxonomy is to help designers decompose their problem into relevant dimensions so they can then select an appropriate approach.

Perhaps a better analysis is to consider each of the key characteristics as an axis in an *n*-dimensional space. The taxonomical classification process then places each application on a point in this *n*-dimensional space.

This is not a precise map. Applications may be complex and thus placement is not exact. The metrics mentioned before delineate architecturally significant boundaries that are not in reality crisp. So, the lines that we have named are somewhat fuzzy. However, an exact position is often not important. Our classification challenge is really only to decide on which side of each boundary our application falls.

In this framework, architectural approaches and the technologies that implement them can be considered to "occupy" some region in this *n*-dimensional space. For instance, a data-centric technology like the Object Management Group (OMG) DDS provides peer-to-peer, fully-redundant connectivity with content filtering. Thus, it would occupy a space that satisfies many reliable, real-time applications with significant numbers of data items, the first three challenge dimensions previously mentioned. The Message Queuing Telemetry Transport (MQTT) protocol, on the other hand, is more suited to the data collection focus challenge. Thus, these technologies occupy different regions of the solution space. Figure 20 represents this concept in three dimensions.

Thus, the application can be placed in the space, and the architectural approaches represented as regions. This reduces the problem of selecting an architecture to one of mapping the application point to appropriate architectural regions.

Of course, this may not be a unique map; the regions overlap. In this case, the process indicates options. The tradeoff is then to find something that fits the key requirements while not imposing too much cost in some other dimension. Thinking of the system as an *n*-dimensional mapping of requirements to architecture offers important clarity and process. It greatly simplifies the search.

TAXONOMY BENEFITS

Defining an IIoT taxonomy will not be trivial. The IIoT encompasses many industries and use cases. It encompasses much more diversity than applications for specialized industry requirements, enterprise IT, or even Consumer IoT. Technologies also evolve quickly, so the scene is constantly shifting. This present state just scratches the surface.

However, the benefit of developing a taxonomical understanding of the IIoT is enormous. Resolving these issues will help system architects choose protocols, network topologies, and compute capabilities. Today, we see designers struggling with issues like server location or configuration, when the right design may not even require servers. Overloaded terms like "real time" and "thing" cause massive confusion between technologies despite the fact that they have no practical use-case overlap. The industry needs a better framework to discuss architectural fit.

Collectively, organizations like the IIC enjoy extensive experience across the breadth of the IIoT. Mapping those experiences to a framework is the first step in the development of a systems science of the IIoT. Accepting this challenge promises to help form the basis of a better understanding and logical approach to designing tomorrow's industrial systems.

STANDARDS AND PROTOCOLS FOR CONNECTIVITY

IoT Protocols

The IoT Protocol Roadmap in Figure 21 outlines the basic needs of the IoT. Devices must communicate with each other (D2D), device data must be collected and sent to the server infrastructure (D2S), and that server infrastructure has to share device data (S2S), possibly providing it back to devices, to analysis programs or to people.

From 30,000 feet, the main IoT protocols can be described in this framework as:

- **MQTT**: A protocol for collecting device data and communicating it to servers (D2S)[10]
- **XMPP**: A protocol best for connecting devices to people, a special case of the D2S pattern, since people are connected to the servers[11]
- **DDS**: A fast bus for integrating intelligent machines (D2D)[12]
- **Advanced Message Queuing Protocol (AMQP)**: A queuing system designed to connect servers to each other (S2S) [13]
- **Open Platform Communications (OPC) Unified Architecture (OPC UA)**: A control plane technology that enables interoperability between devices[14]

Each of these protocols is widely adopted. There are at least 10 implementations of each. Confusion is understandable, because the high-level positioning is similar. In fact, the first four all claim to be real-time publish-subscribe IoT protocols that can connect thousands of things. Worse, those claims are true, depending on how you define "real

10 http://mqtt.org/

11 http://xmpp.org/

12 http://portals.omg.org/dds/

13 https://www.amqp.org/

14 http://opcfoundation.org

time," "publish-subscribe" and "thing."

Nonetheless, all five are very different indeed! They do not, in fact, overlap much at all. Moreover, they don't even fill strictly comparable roles. For instance, OPC UA and DDS are best described as information systems that have a protocol; they are much more than just a way to send bits.

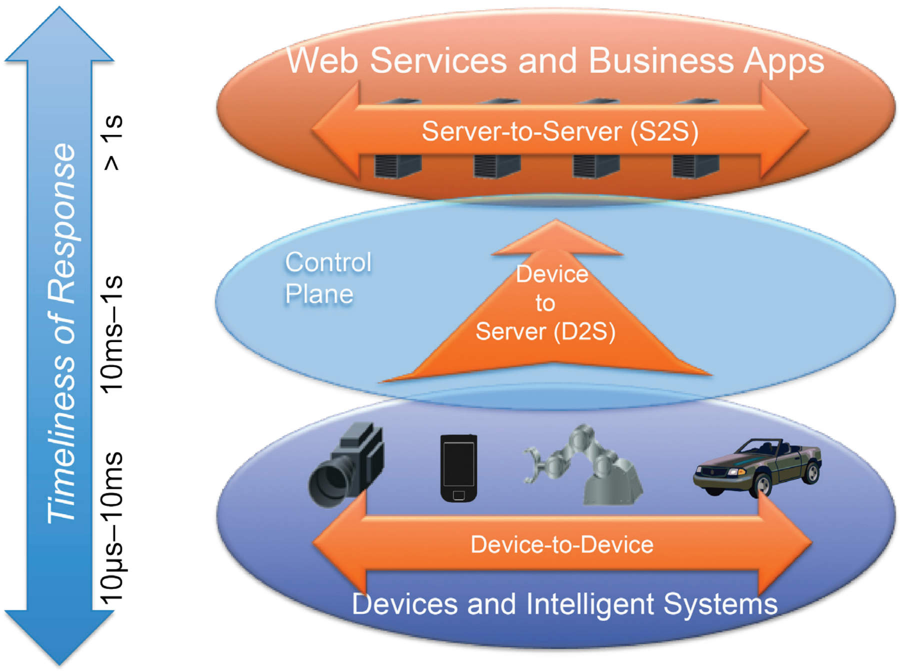


FIGURE 21: IoT protocol roadmap. Devices communicate with each other (D2D) and send data to the IT infrastructure (D2S). The IT infrastructure servers use the data (S2S), communicating back to devices or to people.

The previously mentioned simple taxonomy frames the basic protocol use cases (Figure 21). Of course, it's not really that simple. For instance, the "control plane" represents some of the complexity in controlling and managing all these connections. Many protocols cooperate in this region.

Today's enterprise Internet supports hundreds of protocols; the IoT will support hundreds more. It's important to understand the class of use that each of these important protocols addresses.

MQTT

MQTT targets device data collection (Figure 22). As the name states, the main purpose is telemetry or remote monitoring. Its goal is to collect data from many devices and transport that data to the IT infrastructure. It targets large networks of small devices that need to be monitored or controlled from the cloud.

MQTT makes little attempt to enable device-to-device transfer, nor to "fan-out" the data to many recipients. Since it has a clear, compelling single application, MQTT is simple, offering few control options. It also doesn't need to be particularly fast. In this context, "real time" is typically measured in seconds.

A hub-and-spoke architecture is natural for MQTT. All the devices connect to a data concentrator server. Most applications don't want to lose data regardless of how long retries take, so the protocol works on top of TCP, which provides a simple, reliable stream. Since the IT infrastructure uses the data, the entire system is designed to easily transport data into enterprise technologies like ActiveMQ and ESBs.

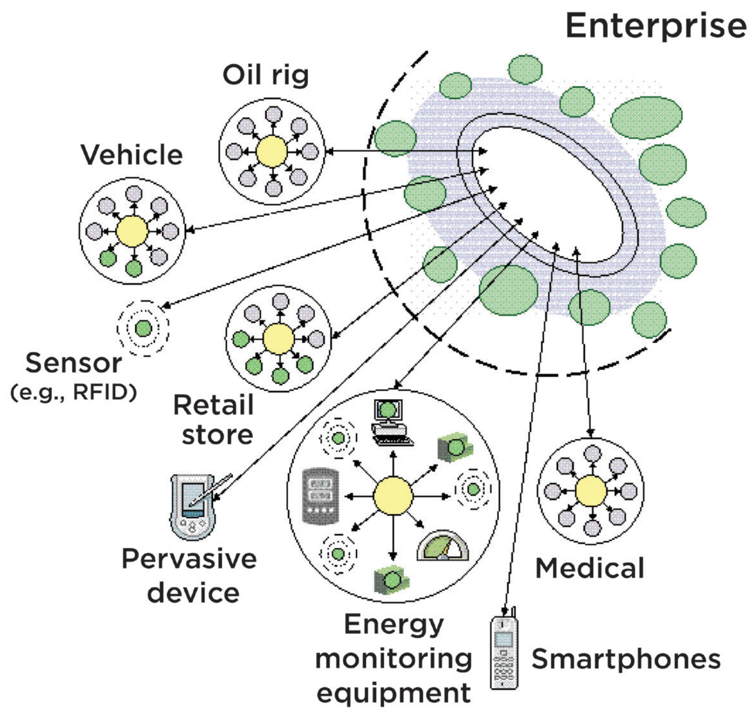


FIGURE 22: Message Queuing Telemetry Transport (MQTT) implements a hub-and-spoke data collection system.

MQTT targets applications like monitoring an oil pipeline for leaks or vandalism. Those thousands of sensors must be concentrated into a single location for analysis. When the system finds a problem, it can take action to correct that problem. Other applications for MQTT include power usage monitoring, lighting control, and even intelligent gardening. They share a need for collecting data from many sources and making it available to the IT infrastructure.

XMPP

XMPP was originally called "Jabber." It was developed for instant messaging (IM) to connect people to other people via text messages (Figure 23). XMPP stands for Extensible Messaging and Presence Protocol. Again, the name belies the targeted use: presence—meaning people are intimately involved.

XMPP uses the XML text format as its native type, making person-to-person communications natural. Like MQTT, it runs over TCP, or perhaps over HTTP on top of TCP. Its key strength is a name@domain.com addressing scheme that helps connect the needles in the huge Internet haystack.

In the IoT context, XMPP offers an easy way to address a device. This is especially handy if that data is going between distant, mostly unrelated points, just like the

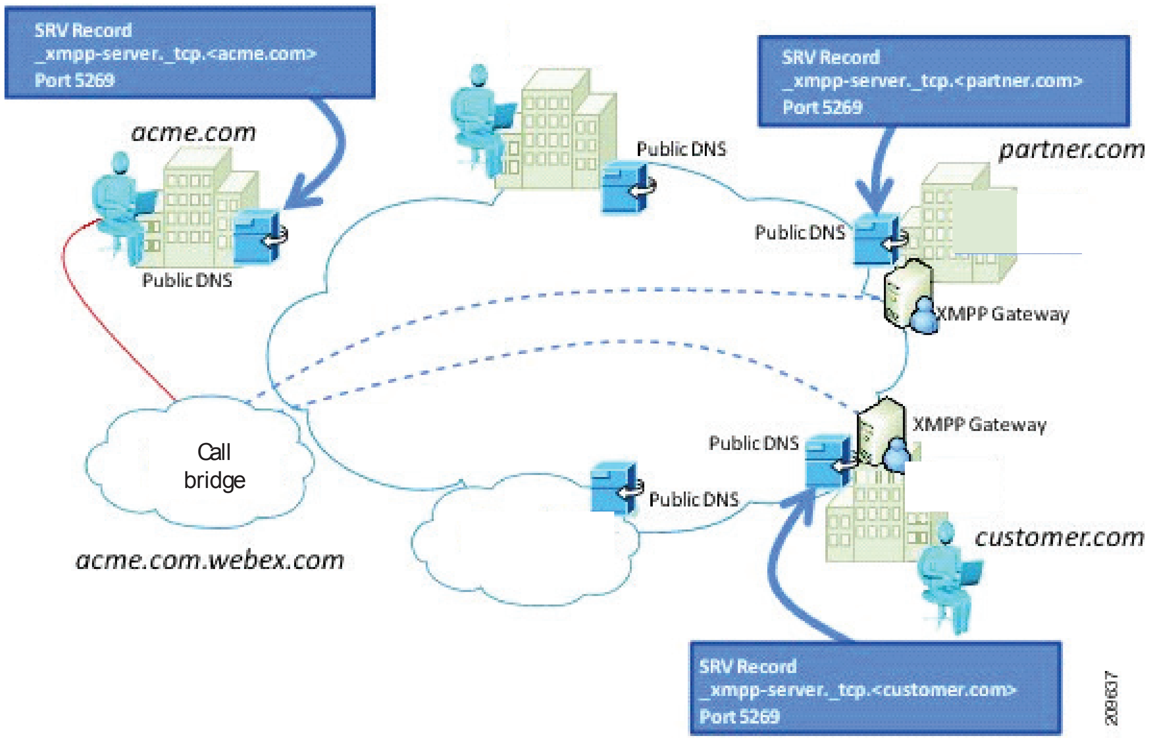


FIGURE 23: Extensible Messaging and Presence Protocol (XMPP) provides text communications between diverse points.

person-to-person case. It's not designed to be fast. In fact, most implementations use polling or checking for updates only on demand. A protocol called Bidirectional-streams over Synchronous HTTP (BOSH) lets servers push messages. But "real time" to XMPP is on a human scale, measured in seconds.

XMPP provides a great way, for instance, to connect your home thermostat to a Web server so you can access it from your phone. Its strengths in addressing security and scalability make it appropriate for Consumer IoT applications.

DDS

In contrast to MQTT and XMPP, DDS targets devices that directly use device data. It distributes data to other devices (Figure 24). DDS's main purpose is to connect devices to other devices. It is a data-centric middleware standard with roots in high-performance defense, industrial, and embedded applications. DDS can efficiently deliver millions of messages per second to many simultaneous receivers.

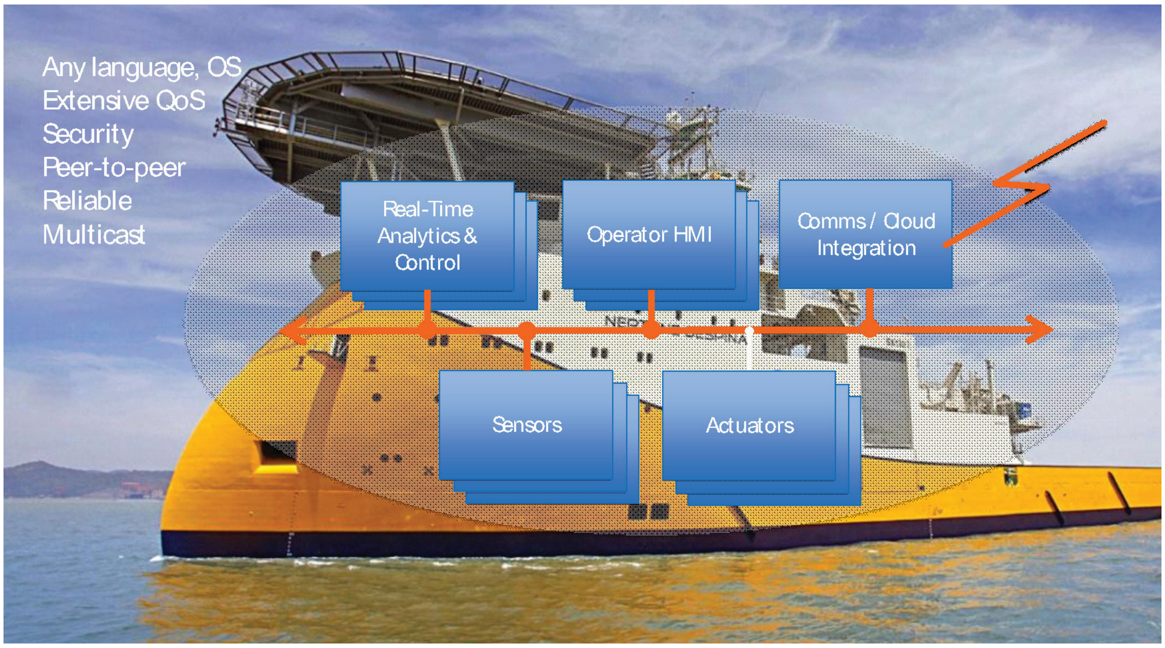


FIGURE 24: Data Distribution Service (DDS) connects devices at physics speeds into a single distributed application

Devices demand data very differently than the IT infrastructure demands data. First, devices are fast. "Real time" may be measured in milliseconds or microseconds. Devices need to communicate with many other devices in complex ways, so TCP's simple and reliable point-to-point streams are far too restrictive. Instead, DDS offers detailed quality-of-service (QoS) control, multicast, configurable reliability and pervasive redundancy. In addition, fan-out is a key strength. DDS offers powerful ways to filter and select exactly which data goes where, and "where" can be thousands of simultaneous destinations. Some devices are small, so there are lightweight versions of DDS that run in constrained environments.

Hub-and-spoke is completely inappropriate for device data use. Rather, DDS implements direct device-to-device "bus" communication with a relational data model. This is often termed a "data bus" because it is the networking analog to a database. Similar to the way a database controls access to stored data, a data bus controls data access and updates by many simultaneous users. This is exactly what many high-performance devices need to work together as a single system.

High-performance integrated device systems use DDS. It is the only technology that delivers the flexibility, reliability, and speed necessary to build complex, real-time applications. DDS is very broadly used. Applications include wind farms, hospital integration, medical imaging, autonomous planes and cars, rail, asset tracking, automotive test, smart cities, communications, data center switches, video sharing, consumer electronics, oil & gas drilling, ships, avionics, broadcast television, air traffic control, SCADA, robotics and defense.

RTI has experience with nearly 1,000 applications. DDS connects devices together into working distributed applications at physics speed.

AMQP

AMQP is sometimes considered an IoT protocol. AMQP is all about queues (Figure 25). It sends transactional messages between servers. As a message-centric middleware that arose from the banking industry, it can process thousands of reliable queued transactions.

AMQP is focused on not losing messages. Communications from the publishers to exchanges and from queues to subscribers use TCP, which provides strictly reliable point-to-point connection. Further, endpoints must acknowledge acceptance of each message. The standard also describes an optional transaction mode with a formal multiphase commit sequence. True to its origins in the

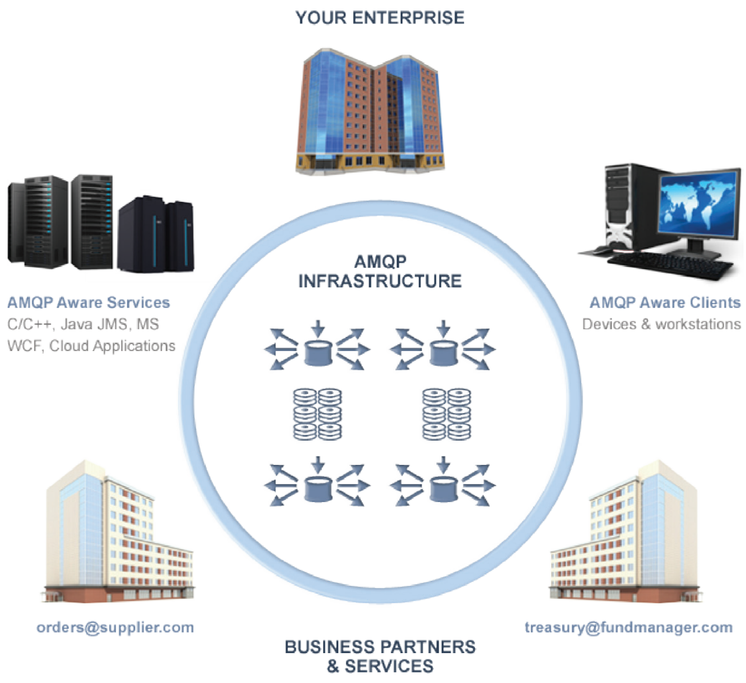


FIGURE 25: Advanced Message Queuing Protocol (AMQP) shares data reliably between servers.

banking industry, AMQP middleware focuses on tracking all messages and ensuring each is delivered as intended, regardless of failures or reboots.

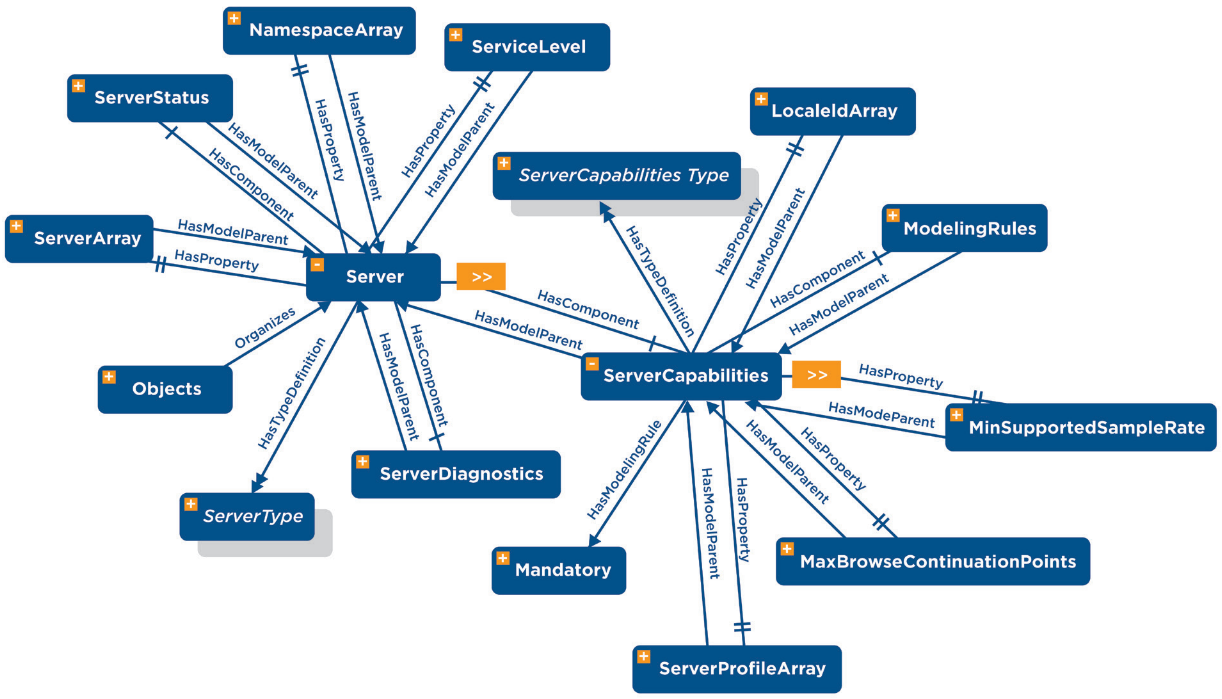
AMQP is mostly used in business messaging. It usually defines "devices" as mobile handsets communicating with back-office data centers. In the IoT context, AMQP is most appropriate for the control plane or server-based analysis functions.

OPC UA

OPC UA is an upgrade of the venerable OPC (OLE for Process Control) protocol. OPC is operational in thousands of factories all over the world. Traditionally, OPC was used to configure and query plant-floor servers (usually Programmable Logic Controllers (PLCs)). Actual device-to-device communications were then effected via a hardware-based "fieldbus" such as ModBus or PROFINET.

OPC UA retains some of that flavor; it connects and configures plant-floor servers. The UA version adds better modeling capabilities. Thus, a remote client (e.g., a graphical interface) can "browse" the device data controlled by a server on the floor. By allowing this introspection across many servers, clients can build a model of the "address space" of all the devices on the floor.

FIGURE 26: OPC UA allows applications to browse a system's object model and interpret the devices and their connections



OPC UA specifically targets manufacturing. It connects applications at the shop-floor level as well as between the shop floor and the enterprise IT cloud. In the taxonomy previously mentioned, it targets systems that require runtime integration.

The Bottom Line: How to Choose?

The IoT needs many protocols. Those outlined here differ markedly. Perhaps it's easiest to categorize them along a few key dimensions: QoS, addressing and application.

QoS control is a much better metric than the overloaded "real-time" term. QoS control refers to the flexibility of data delivery. A system with complex QoS control may be harder to understand and program, but it can build much more demanding applications.

For example, consider the reliability QoS. Most protocols run on top of TCP, which delivers strict, simple reliability. Every byte put into the pipe must be delivered to the other end, even if it takes many retries. This is simple and handles many common cases, but it doesn't allow timing control. TCP's single-lane traffic backs up if there's a slow consumer.

Because it targets device-to-device communications, DDS differs markedly from the other protocols in QoS control. In addition to reliability, DDS offers QoS control of "liveliness" (when you discover problems), resource usage, discovery and even timing.

Next, "discovery"—finding the data needle in the huge IoT haystack—is a fundamental challenge. XMPP shines for "single item" discovery. Its "user@domain" addressing leverages the Internet's well-established conventions. However, XMPP doesn't easily handle large data sets connected to one server. With its collection-to-a-server design, MQTT handles that case well. If you can connect to the server, you're on the network. AMQP queues act similarly to servers, but for S2S systems. Again, DDS is an outlier. Instead of a server, it uses a background "discovery" protocol that automatically finds data. DDS systems are typically more contained; discovery across the wide-area network (WAN) or huge device sets requires special consideration.

OPC UA specializes in communicating the information about the system, its configuration, topology and data context (the “metadata”). These are exposed in the collective address space of the individual OPC UA servers. This data can be accessed by OPC UA clients; they can see what is available and choose what to access. OPC UA is not designed for flexible device-to-device interaction.

Perhaps the most critical distinction comes down to the intended applications. Inter-device data use is a fundamentally different use case from device data collection. For example, turning on your light switch remotely (best for XMPP) is worlds apart from generating that power (DDS), monitoring the transmission lines (MQTT) or analyzing the power usage back at the data center (AMQP).

Of course, there is still confusion. For instance, DDS can serve and receive data from the cloud, and MQTT can send information back out to devices. Nonetheless, the fundamental goals of all five protocols differ, the architectures differ, and the capabilities differ. All of these protocols are critical to the (rapid) evolution of the IoT. And all have a place; the IIoT is a big place with room for many protocols. To make confusion even worse, many applications integrate many subsystems, each with different characteristics.

With this variety, what’s the best path? Experience suggests that designers first identify the application’s toughest challenge, and then choose the technology that best meets that single challenge. This is a critical decision; choose carefully and without prejudice of what you know. The aforementioned requirements-based dimensional decomposition can help. After this step, the choice is usually fairly obvious; most applications contain a key challenge that clearly fits better with one or the other.

Once the hardest challenge is met, the best way to cover the rest of the application is usually to push your initial choice as far as it will go. There are many bridging technologies, so it is possible to mix technologies. However, it’s usually easier to avoid multiple protocol-integration steps when possible.

In the long term, the technologies will offer better interoperability. The IIC’s “connectivity core standard” design described later is a key approach. So, regardless of your initial choice, the vendor communities are working to provide a non-proprietary path to interoperability.

CONNECTIVITY ARCHITECTURE FOR THE IIOT

There is no way to build large distributed systems without connectivity. Enterprise and human-centric communications are too slow or too sparse to put together large networks of fast devices. These new types of intelligent machines need a new technology. That technology has to find the right data and then get that data where it needs to go on time. It has to be reliable, flexible, fast and secure. Perhaps not as obviously, it also must work across many types of industries. Only then can it enable the efficiencies of common machine-based and cloud-based infrastructure for the IIoT.

Connectivity faces two key challenges in the IIoT: interoperability and security. Interoperability is a challenge because the IIoT must integrate many subsystems with different designs, vendor equipment, or legacy infrastructures. Security is a challenge because most enterprise security approaches target hub-and-spoke designs with a natural center of trust. Those designs cannot handle vast networks of devices that must somehow trust each other.

The IIC's IIRA addresses both. Ultimately, the IIoT is about building distributed systems. Connecting all the parts intelligently so the system can perform, scale, evolve, and function optimally is the crux of the IIRA. The IIoT must integrate many standards and connectivity technologies. The IIC architecture explicitly blends the various connectivity technologies into an interconnected future that can enable the sweeping vision of a hugely connected new world.

The *n*-Squared Challenge

When you connect many different systems, the fundamental problem is the "n-squared" interconnect issue. Connecting two systems requires matching many aspects, including protocol, data model, communication pattern and QoS parameters like reliability, data rate or timing deadlines. While connecting two systems is a challenge, it is solvable with a special-purpose "bridge." But that approach doesn't scale; connecting *n* systems together requires n-squared bridges. As *n* gets large, this becomes daunting.

One way to ease this problem is to keep *n* small. You can do that by dictating all standards and technologies across all systems that interoperate. Many industry-specific standards bodies successfully take this path. For instance, the European Generic Vehicle Architecture (GVA) specifies every aspect of how to build military ground vehicles, from low-level connectors to top-level data models. The German Industrie 4.0 effort takes a similar pass at the manufacturing industry, making choices for ordering and delivery, factory design, technology, and product planning. Only one standard per task is allowed.

This approach eases interoperation. Unfortunately, the result is limited in scope because the rigidly-chosen standards cannot provide all functions and features. There are simply too many special requirements to effectively cross industries this way. Dictating standards also doesn't address the legacy integration problem. These two restrictions (scope and legacy limits) make this approach unsuited to building a wide-ranging, cross-industry Industrial Internet.

On the other end of the spectrum, you can build a very general bridge point. Enterprise web services work this way, using an Enterprise Service Bus (ESB), or a mediation bus like Apache Camel. However, despite the "bus" in its name, an ESB is not a distributed concept. All systems must connect to a single point, where each incoming standard is mapped to a common object format. Because everything maps to one format, the ESB requires only one-way translation, avoiding the n-squared problem. Camel, for instance, supports hundreds of adapters that each convert one protocol or data source to and from Camel's internal object format. Thus, any protocol can in principal connect to any other.

Unfortunately, this doesn't work well for demanding industrial systems. The single ESB service is an obvious choke and failure point. ESBs are large, slow programs. In the enterprise, ESBs connect large-grained systems executing only a few transactions per second. Industrial applications need much faster, reliable, smaller-grained service. So, ESBs are not viable for most IIoT uses.

The IIRA Connectivity Core Standard

The IIRA takes an intermediate approach.[15] The design introduces the concept of a "Connectivity Core Standard." Unlike an ESB, the core standard is very much a distributed concept. Some endpoints can connect directly to the core standard. Other endpoints and subsystems connect through "gateways." The core standard then connects them all together. This allows multiple protocols without having to bridge between all possible pairs. Each needs only one bridge to the core.

15 http://www.iiconsortium.org/IIRA.htm

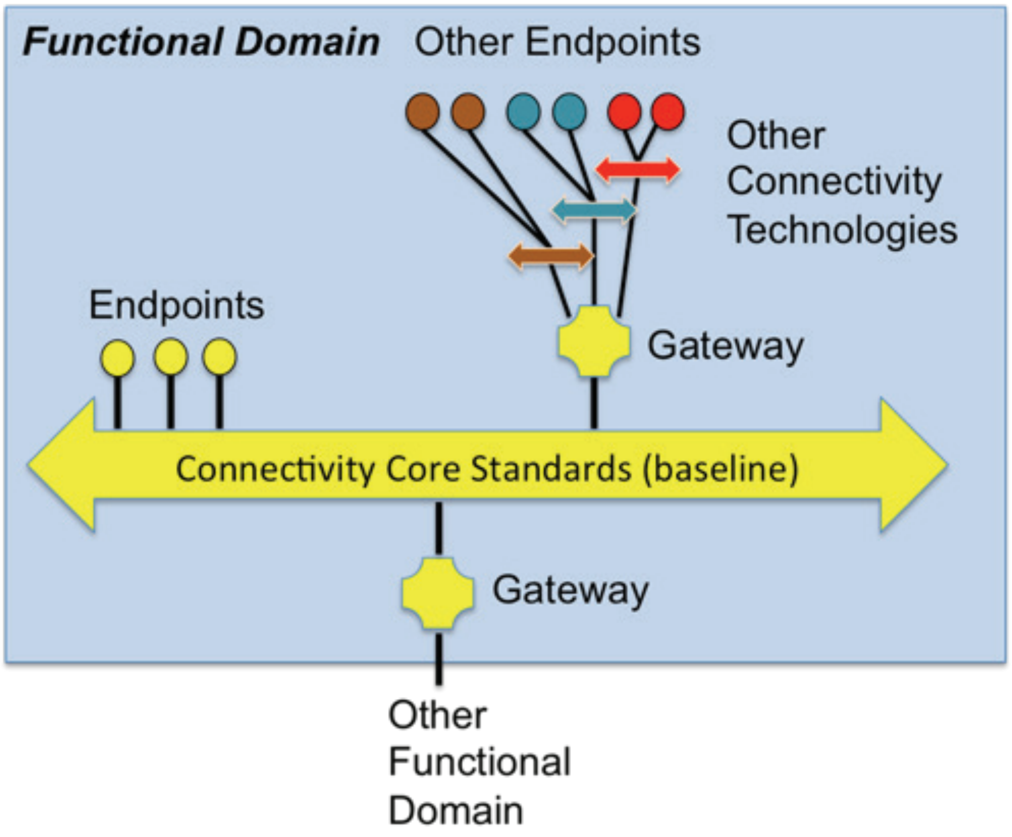


FIGURE 27: The IIRA connectivity architecture specifies a QoS-controlled, secure connectivity core standard. All other connectivity standards must only bridge to this one core standard.

Like an ESB, this solves the n-squared problem. But, unlike an ESB, it provides a fast, distributed core, replacing the centralized service model. Legacy and less-capable connectivity technologies transform through a gateway to the core standard. There are only *n* transformations, where *n* is the number of connectivity standards.

Obviously, this design requires a very functional connectivity core standard. Some systems may get by with slow or simple cores. But most industrial systems need to identify, describe, find and communicate a lot of data with demands unseen in other contexts. Many applications need delivery in microseconds or the ability to scale to thousands or even millions of data values and nodes. The consequences of a reliability failure can be severe. Since the core standard really is the core of the system, it has to perform.

The IIRA specifies the key functions that connectivity framework and its core standard should provide: data discovery, exchange patterns and QoS. QoS parameters include delivery reliability, ordering, durability, lifespan and fault tolerance functions. With these capabilities the core connectivity can implement the reliable, high-speed, secure transport required by demanding applications across industries.

The IIRA outlines several data QoS capabilities for the connectivity core standard. These ensure efficient, reliable, secure operation for critical infrastructure.

Data Quality of Service (QoS)

1. **Delivery:** Provide reliability and redelivery
2. **Timeliness:** Prioritize and inform when information is “late”
3. **Ordering:** Deliver in the order produced or received
4. **Durability:** Support late joiners, survive failures
5. **Lifespan:** Expire stale information
6. **Fault Tolerance:** Enable redundancy and failover
7. **Security:** Ensure confidentiality, integrity, authenticity and non-repudiation

The IIoT Approach to Security

Security is also critical. To make security work correctly, it must be intimately married to the architecture. For instance, the core standard may support various patterns and delivery capabilities. The security design must match those exactly. For example, if the connectivity supports publish-subscribe, so must security. If the core supports multicast, so must security. If the core supports dynamic plug-and-play discovery, so must security. Security that is intimately married to the architecture can be imposed at any time without changing the code. Security becomes just another controlled QoS, albeit more complexly configured. This is a very powerful concept.

The integrated security must extend beyond the core. The IIRA allows for that too; all other connectivity technologies can be secured at the gateways.

DDS as a Core Standard

The IIRA does not currently specify standards; the IIC will take that step in the next release. However, it's clear that the DDS standard is a great fit to the IIRA for many applications. DDS provides automated discovery, each of the patterns specified in the IIRA, all the QoS settings and intimately integrated security.

This is no accident. The IIRA connectivity design draws heavily on industry experience with DDS. DDS has thousands of successful applications in power systems (huge hydropower dams, wind farms, microgrids), medicine (imaging, patient monitoring, emergency medical systems), transportation (air traffic control, vehicle control, automotive testing), industrial control (SCADA, mining systems, PLC communications) and defense (ships, avionics, autonomous vehicles). The lessons learned in these applications were instrumental in the design of the IIRA.

DDS UNIQUE FEATURES

DDS is not like other middleware. It directly addresses real-time systems. It features extensive fine control of real-time QoS parameters, including reliability, bandwidth control, delivery deadlines, liveliness status, resource limits and (new) security. It explicitly manages the communication "data models," or types, used to communicate between endpoints. It is thus a "data-centric" technology. Like a database, which provides data-centric storage, DDS understands the contents of the information it manages. DDS is all about the data. This data-centric nature, analogous to a database, justifies the term "databus."

At its core, DDS implements a connectionless data model with the ability to communicate data with the desired QoS. Originally, DDS focused on publish-subscribe communications; participants were either publishers of data or subscribers to data. Later versions of the specification added request-reply as a standard pattern. Currently, RTI also offers a full queuing service that can implement "one of *n*" patterns for applications like load balancing. The key difference between DDS and other approaches is not the publish-subscribe pattern. The key difference is data-centricity.

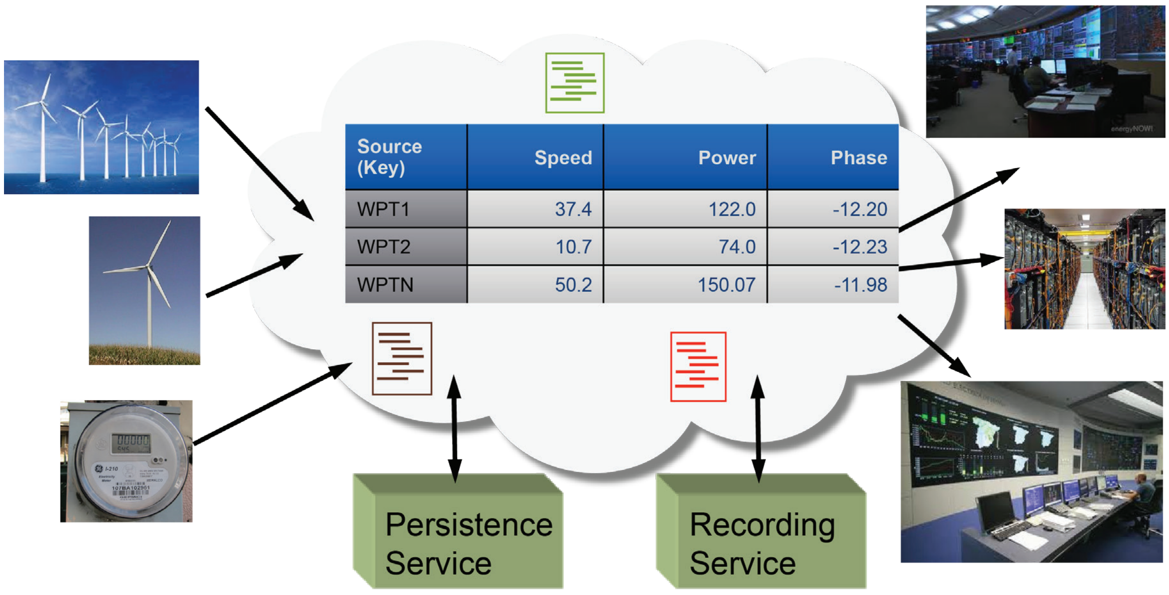
A DDS-based system has no hard-coded interactions between applications. The data bus automatically discovers and connects publishing and subscribing applications. No configuration changes are required to add a new smart machine to the network. The data bus matches and enforces QoS.

DDS overcomes problems associated with point-to-point system integration, such as lack of scalability, interoperability, and the ability to evolve the architecture. It enables plug-and-play simplicity, scalability and exceptionally high performance.

The DDS Databus

The core architecture is a "data bus" that ties all the components together with strictly controlled data sharing. The infrastructure implements full QoS control over reliability, multicast, security and timing. It supports fully redundant sources, sinks, networks, and services to ensure highly reliable operation. It needs no communication servers for discovery or configuration; instead, it connects data sources and sinks through a background "meta traffic" system that supports massive scale with no servers.

The data bus technology scales across millions of data paths, ensures ultra-reliable operation, and simplifies application code. It does not require servers, greatly easing configuration and operations while eliminating failure and choke points. DDS is by far the most proven technology



Source (Key)	Speed	Power	Phase
WPT1	37.4	122.0	-12.20
WPT2	10.7	74.0	-12.23
WPTN	50.2	150.07	-11.98

for reliable, high-performance, large-scale IIoT systems. As of this writing, there are at least 12 implementations. Several of these are backed by commercial vendors. RTI provides the leading implementation.

Conceptually, DDS is simple. Distributed systems must share information. Most middleware works by simply sending that information to others as messages.

DDS sends information too, of course. However, DDS conceptually has a local store of data, which looks to the application like a simple database table. When you write, it goes into your local store, and then the messages update the appropriate stores on remote nodes. When you read, you just read locally. The local stores together give the applications the illusion of a global data store. Importantly, this is only an illusion. There is no global place where all the data lives; that would be a database. In a data bus, each application stores locally only what it needs and only for as long as it needs it. Thus, DDS deals with data in motion; the global data store is a virtual concept that in reality is only a collection of transient local stores.

DDS also matches producers and consumers to ensure proper operation. The matching includes data flow rates, remote "liveliness," filtering, security and performance. Once matched, the middleware enforces the "contract." For instance, if a subscriber needs updates 1,000 times per second, the middleware will ensure a fast producer is available. The standard defines more than 20 of these QoS policies.

FIGURE 28: Data-Centric Communications. The data bus links any language, device or transport. It automatically discovers information sources, understands data types and communicates them to interested participants. It scales across millions of data paths, enforces sub-millisecond timing, ensures reliability, supports redundancy and selectively filters information. Each path can be unicast, multicast, open data or fully secure. In the figure a medical device that produces heart waveforms will send only one patient's information to the nursing station, at a rate it can handle, if it has permission to receive the information.

DATA-CENTRICITY MAKES DDS DIFFERENT

Systems are all about the data. Distributed systems must also share and manage that data across many processors and applications. The strategy to understand and manage this state is a fundamental design decision.

Data-centricity can be defined by these properties:

- The interface is the data. There are no artificial wrappers or blockers to that interface like messages, objects, files, or access patterns.
- The infrastructure understands that data. This enables filtering/searching, tools, and selectivity. It decouples applications from the data and thereby removes much of the complexity from the applications.
- The system manages the data and imposes rules on how applications exchange data. This provides a notion of "truth." It enables data lifetimes, data model matching, CRUD interfaces, etc.

An analogy with the database, a data-centric storage technology, is instructive. Before databases, storage systems were files with application-defined (ad hoc) structure. A database is also a file, but it's a very special file. A database has known structure and access control. A database defines "truth" for the system; data in the database can't be corrupted or lost.

By enforcing structure and simple rules that control the data model, databases ensure consistency. By exposing the structure to all users, databases greatly ease system integration. By allowing discovery of data and schema, databases also enable generic tools for monitoring, measuring, and mining information.

Like a database, data-centric middleware imposes known structure on the transmitted data. The data bus also sends messages, but it sends very special messages. It sends only messages specifically needed to maintain state. Clear rules govern access to the data, how data in the system changes and when participants get updates. Importantly,

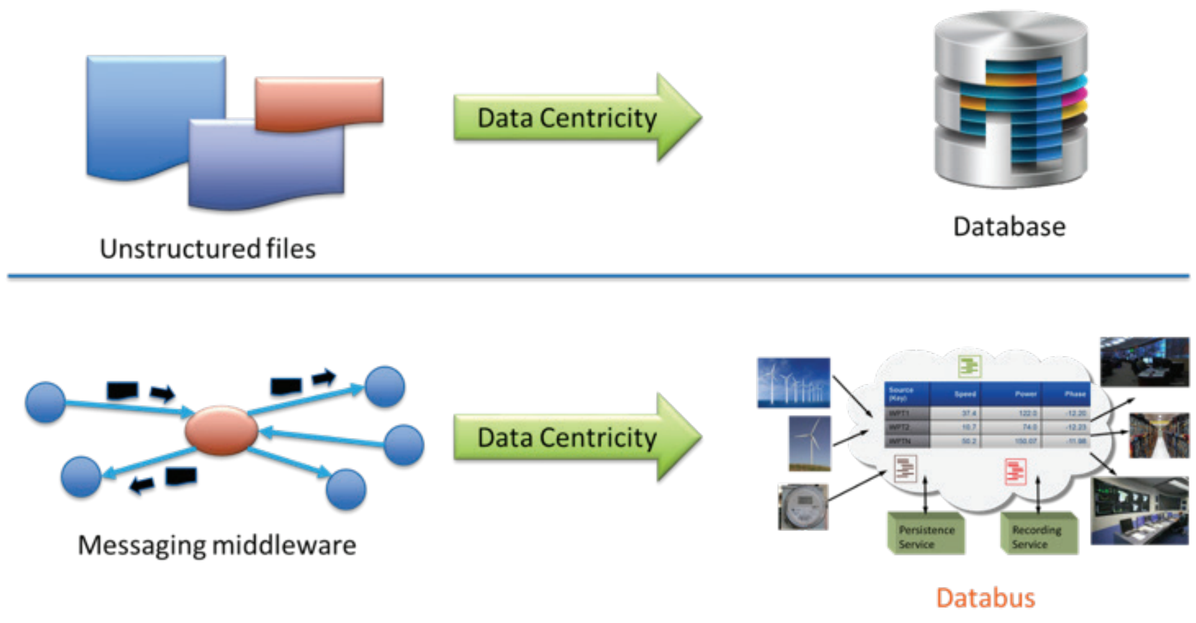


Figure 29: Data-centric middleware does for data in motion what a database does for data at rest. The database's data-centric storage fundamentally enables the simplified development of very complex information systems. Analogously, the data bus offers data-centric networking that fundamentally enables the simplified development of very complex distributed systems. Both move much of the complexity from the application (user code) to the infrastructure.

the infrastructure sends messages. To the applications, the system looks like a controlled global data space. Applications interact directly with data and data properties like age and rate. There is no application-level awareness or concept of "message."

With knowledge of the structure and demands on data, the infrastructure can do things like filter information, selecting when or whether to do updates. The infrastructure itself can control QoS like update rate, reliability and guaranteed notification of peer liveliness. The infrastructure can discover data flows and offer those to applications and generic tools alike. This knowledge of data status, in a distributed system, is a crisp definition of "truth." As in databases, this accessible source of truth greatly eases system integration. The structure also enables tools and services that monitor and view information flow, route messages and manage caching.

THE FUTURE OF THE IIOT

The IIoT is clearly in its infancy. Like the early days of the Internet, the most important IIoT applications are not yet envisioned. The "killer application" that drove the first machine-to-machine connections for the Internet was email. However, once connected, the real power of distributed systems created an entirely new ecosystem of value. This included web pages, search, social media, online retail and banking, and so much more. The real power of the Internet was barely hinted in its early days.

The IIoT will likely follow a similar pattern. Today, many companies are most focused on collecting data from industrial systems and delivering it to the cloud for analysis. This is important for predictive maintenance, system optimization, and business intelligence. This "killer app" is driving the initial efforts to build connected systems.

However, the future holds much more promise than optimizing current systems. By combining high-quality connectivity with smart learning and machine intelligence, the IIoT future holds many new systems that will revolutionize our world. It will, for instance, save hundred of thousands of lives a year in hospitals, make renewable energy sources truly practical, and completely transform daily transportation. Projections of the economic and social benefits range greatly, but all agree the impact is measured in the multiple trillions of dollars in a short 10 years. That is a daunting, but inspiring number. The IIoT will be a daunting, but inspiring transformation across the face of industry.

ABOUT RTI

RTI provides the connectivity platform for the Industrial Internet of Things.

Our RTI Connext® messaging software forms the core nervous system for smart, distributed applications. RTI Connext allows devices to intelligently share information and work together as one integrated system. RTI was named "The Most in Influential Industrial Internet of Things Company" in 2014 by Appinions and published in Forbes.

Our customers span the breadth of the Internet of Things, including medical, energy, mining, air tra c control, trading, automotive, unmanned systems, industrial SCADA, naval systems, air and missile defense, ground stations, and science.

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, privately held and headquartered in Sunnyvale, California.

rti

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