WHITEPAPER

The Layered Databus Architecture

USING A DATA-CENTRIC APPROACH TO BRIDGING LEGACY SYSTEMS WITH MODERN SYSTEM DESIGN

EXECUTIVE SUMMARY

In the industrial world, the pure greenfield development project is a rare thing. In most cases, the greenfield portion of a new project must still interoperate with a legacy system or even multiple legacy systems. Today's data-driven operations rely on interconnected systems representing a mix of both proprietary and open technology.

It is becoming more common for legacy systems to have to expand or scale with non-traditional systems. An example would be autonomous vehicle sensors and software working with an internal legacy CAN bus, or a completely autonomous vehicle interacting with road infrastructure elements that include a mix of legacy and smart transportation elements. The design practices that were and are still used to maintain and operate legacy systems cannot be discarded. Some legacy systems are difficult to modernize—the code upon which they were built is decades old. It's simply not practical for engineering teams to create modern data-driven platforms using legacy design practices. At the same time, engineers must preserve the integrity of legacy systems that are based on proprietary code. This can be costly: In virtually every industrial system today, integration is the most expensive part of expansion.

The software framework that bridges from legacy to modern systems is the data-centric databus built on the Data Distribution Service™ (DDS) standard. The DDS standard has been used in thousands of systems to solve increasingly complex design integration challenges without the need for custom coding. Engineers can create multiple (even hundreds) of DDS-based layers (databuses) to separate, isolate and selectively share communications. It can do this with an impressive and flexible set of mechanisms, notably Quality of Service (QoS), filtering, automatic discovery and security.

This paper covers the basics of a layered databus, including the capabilities of layers, defining layers, connecting layers and decomposing layers. It also provides use cases from different industries on how the layered databus can be used to address legacy interoperability requirements without incurring the penalties of legacy design practices.

STRAINS AND CONSTRAINTS OF APPLYING LEGACY DESIGN TO GREENFIELD SYSTEMS

With rare exception, most new applications in the industrial world must integrate with legacy systems. Realistically, legacy design practices are too cumbersome to support the rapid expansion and scalability of digital operations. Virtually every industry—aerospace, defense, healthcare, transportation, manufacturing, oil and gas, utilities and others—is confronted by the same problem:

> *How to create a communications platform that enables legacy systems to scale to meet the proliferating demands of the modern industrial network.*

If engineers choose to use legacy design practices to respond to these needs, there are significant penalties as illustrated in Figure 1.

THE DIFFERENCE BETWEEN A DATABASE AND A DATABUS

A database and a databus both handle structured data that is either written or read, but there are key differences. The data in the database is 'at rest'. Unless it is being deposited or retrieved, database information doesn't go anywhere. In contrast, the data that is handled by a databus is always in motion, making it more suitable for systems requiring scalability and real time or near real-time information in order to operate as designed. It is important to note that a databus isn't a competitive technology to a database; rather, both play a role in handling massive amounts of data, as is illustrated in Figure 2.

Strained resources	Constrained scalability

• The cost of development for a proprietary system is substantial compared to using a standards-based platform.
• With proprietary design practices, companies and institutions may be dependent on a small number of vendors (or even a single vendor) to design and build any major extensions.
• Documentation for proprietary systems is notoriously poor, making it difficult to transition responsibilities for systems development to other vendors, new staff or internal development teams.
• Complex, proprietary systems increase dependence on a few knowledgeable people in the organization; bringing new people up to speed can require extensive training.
• The time to complete a project is likely to be months and even years, depending on the complexity of the connected systems.
• There may be a high failure rate associated with these projects, or an inability to meet all of the project goals.
• Complex programming translates into high maintenance costs.
• The constant demand for new connections puts additional strain on design, development and testing teams.
	• Depending on the diversity of applications, devices and systems that need to be interconnected, the entire design process may be repeated over and over again. For example,
	  • Some systems that are connected may require fault tolerance, while others require low latency. With proprietary development, these differences may send each system to a different development team to address its unique requirements, with a limited ability to reuse code between the two teams.
	• Proprietary systems tend to be one-to-one undertakings, not plug-and-play platforms. For example:
	  • A majority of proprietary systems use sockets for communications; adding a new piece of hardware or application requires custom building a new socket. In addition, these sockets are single purpose and have no ability to apply other functions such as QoS, data filtering or enhanced security.
	• Proprietary design methodologies and monolithic architectures often create single points of failure and a multitude of security gaps as the system is scaled and extended.

Figure 1: Challenges of legacy design practices in modern system architecture.

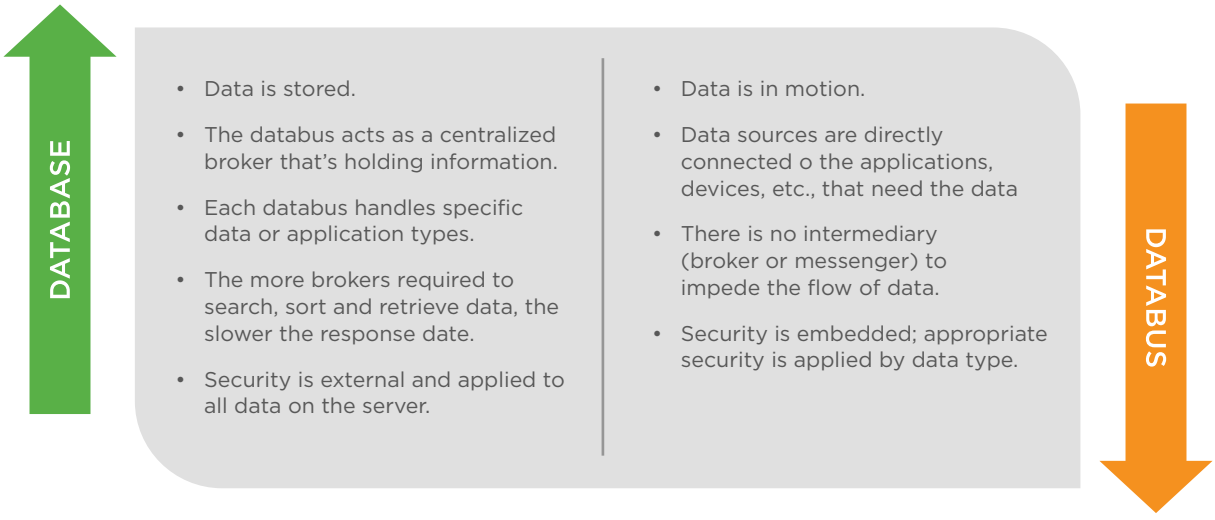


Figure 2: The database and the databus both play a role in optimizing data management in modern industrial systems.

THE DDS DATABUS

The Object Management Group® publishes and maintains the family of specifications based on the Data Distribution Service™ (DDS) standard. DDS is a communications protocol; the DDS communications layer is based on a software databus that runs via a publish/subscribe model. The DDS databus is essentially a shared global space where data is continuously flowing to and from its intended and authorized recipient(s), multiplied by the hundreds, thousands or millions of publishers or subscribers. Following are just some of the notable characteristics of a databus:

- Data-centricity: This refers to data awareness that enables things like intelligent filtering of data and delivering data to the right location at the right time.
- Built-in QoS: QoS parameters can include such information as the rate at which the samples or data will be republished, the reliability of the delivery and the security of data flow.
- Shared data model: The shared data model eliminates data brokers or intermediaries and leads to scalability and expandability in multi-layered databuses. Applications can directly read and write data values, eliminating bottlenecks and contributing to fast, efficient communications. Data samples are cached locally within the readers and writers. Key advantages are: Designers can define all of the topics in their systems, without writing any code; designers can use any data model they prefer; data models are consistent throughout every layer as well as the application life cycle.
- Automatic discovery: A new application or device joining the databus can be provided with previously published state data.
- Integrated system: All applications, devices, modules and subsystems work together as one integrated system.
- Security: The DDS standard includes a plug-in security architecture that connects to a DDS library. The library includes numerous mechanisms, such as: discovery authentication; data-centric access control; cryptography; data tagging and logging; non-repudiation; and secure multicasting.

Note: In describing the databus, "layer" and "domain" are often used interchangeably. In particular, "domain participants" is the description used throughout this paper (and in the DDS standard) to encompass all types of interconnected "things," such as sensors, devices, applications, endpoints, users, systems, etc.

THE LAYERED DDS DATABUS

When there is more than one DDS databus in a design—and there is a need to share information across those databuses—the layered databus comes into play. A layered databus is a multi-domain databus architecture where different domains are served by their own databuses, with the ability to interconnect and enable communications between layers. This provides a common or unifying communications architecture across distributed and/or disparate systems. This architecture provides low-latency, secure, peer-to-peer data communications across logical layers of the system.

The information that is shared between databuses is selective and is based on the DDS publish/subscribe method. That is, the designer chooses the data that one layer requires from another. Not all information is relevant or important.

A layered databus can greatly simplify potential complexity. Specifically, designers are often looking for a way to achieve scalability in order to support hundreds of thousands of devices and applications without additional complexity. Other benefits of layered databuses include:

- Fast device-to-device integration with delivery times in milliseconds or microseconds
- Automatic data and application discovery with and between databuses
- Scalable integration comprising hundreds of thousands of domain participants
- Natural redundancy allowing extreme availability and resilience
- Hierarchical subsystem isolation enabling development of complex systems design
- Security that is customizable via any standard API and runs over any type of transport

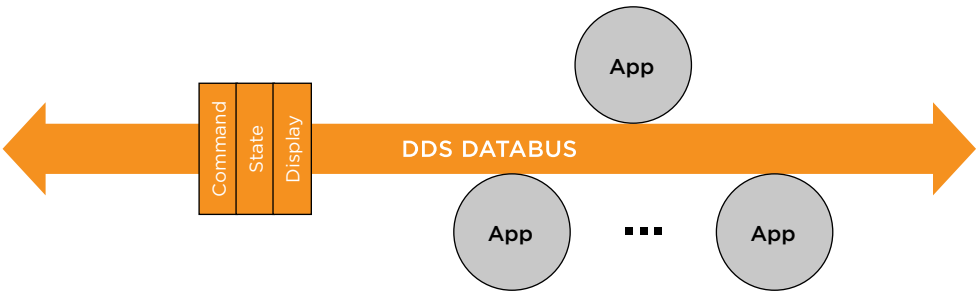


Figure 3: A single DDS databus layer might be defined by its function, such as command and control. Layers can also be defined by a physical location, communication protocol, behavior or other characteristics.

In most modern industrial systems, applications need to share information with other applications in order to perform as designed. The Industrial Internet Consortium® recommends the layered databus as an architectural communications pattern for industrial IoT applications in its Industrial Internet Reference Architecture (IIRA)[1]. The IIRA is a standards-based architectural guideline for designing IIoT systems based on a common framework.

In industrial systems, one common architecture pattern is made up of multiple databuses layered by communication QoS and data model needs. Typically, databuses will be implemented at the edge of a system, which typically represents data collected from sensors or devices used in running smart machines or lower level subsystems as part of a complex system (e.g., a car, an oil rig or a hospital).

On another hierarchy level, there can be one or more databuses that integrate additional machines or systems, facilitating data communications between and within the higher-level control center or backend systems. The backend or control center layer could be the highest layer databus in the system, but there can be more than these three layers. Many layers are possible, as discussed in this white paper.

BUILDING BLOCKS OF A LAYERED DATABUS

This section covers the building blocks of a layered databus. It is important to note that the DDS databus provides flexibility in design choices. For each of the architectural elements discussed here, there may be compelling reasons for a different approach in a particular use case design.

Defining the layers

It is certainly possible to have a design in which all information is localized, requiring only a single DDS databus. But for more complex systems—such as a connected hospital, or an autonomous vehicle—the layered databus architecture is necessary.

A layered databus enables the separation and securing of different domains. In fact, isolating data is a critical component of a layered databus architecture, while ensuring that the relevant information is shared across layers.

There is also the benefit of consistency while scaling. That is, each layer has the same design and features of the DDS databus, which eliminates the need to recreate a new databus from scratch for each layer.

The first step in interconnecting layers is to define them. There are many options for designating different layers, including:

- Domains, identified by a unique integer value known as a domain ID.
- Partitions, to control which DataWriters will match (and communicate with) which DataReaders.
- Multicast groups, for data transfer between applications that are running on the same node.
- Function or purpose, such as a supervisory layer or analytics layer.
- Scope of interest, which pertains to the responsibilities and interests of whoever or whatever is sending and receiving data.
- Location, such as a patient monitor, an autonomous vehicle, an oil rig, etc.
- Transport type, such as UDPv4 or TCPv4.

Connecting the layers

Each layer must have a logical connection to the other, as shown in Figure 4. This connection is sometimes referred to as a bridge or gateway. A single gateway can route data to a single layer or multiple layers. Because the DDS databus standard ensures that the APIs are consistent, designers can use their own bridging technology to connect layers.

The RTI gateway (called the RTI Routing Service) provides comprehensive services while giving designers tremendous flexibility in how these services are applied. Here are some examples:

QoS: Data that flows within and between layers are configured via QoS settings that define how data is delivered between a publisher and subscriber. In the DDS standard, these data flows are called topics. This is a user-configurable parameter, so designers can set individual topics for specific ranges.

Filtering: This is a mechanism that enables designers to select data that meets a certain criteria, e.g., above or below a specific number such as temperature or density. Data that meets the required identified parameters will be shared with its subscribers. Data that does not meet the parameters will not be shared. For example, in a smart factory, only data that is outside the normal ranges may be passed along to the maintenance department, as an indication that a machine is about to malfunction. One of the benefits of the DDS databus and RTI Routing Service is that data can be selectively filtered based on the contents of the data. That provides designers with highly-granular resolution of data separation, meaning the ability to separate individual pieces of data based on specific values (at either the source or destination).

Automatic discovery: Discovery is automatic, but also controlled. Designers can choose automated discovery between Layer 1 and Layer 2, but not with participants in Layer 3. Discovery can even be fine-tuned to the degree of discovery that is automated.

Interoperability of the layers

With the DDS databus standard, designers have more than one option for achieving interoperability, for example:

- Translation for modification of information, perhaps to create a common way to represent diverse information (e.g., translating temperature from Fahrenheit to Celsius)
- Adaptation to connect different technologies (e.g., connecting DDS to MQTT or the distributed interactive simulation protocol [DIS])
- Shims for supporting an old API in a new environment or a new API in an older environment

Decomposing the layers

How many layers are needed for a design? Is there a minimum or maximum number?

[1] Industrial Internet Consortium

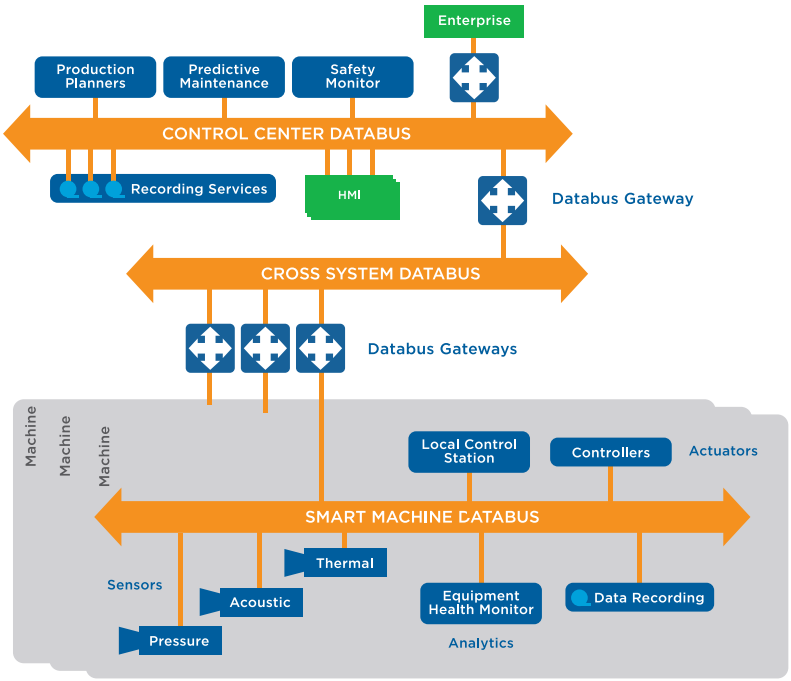


Figure 4: A simple view of a layered architecture connected by a gateway.

This is a question that should be considered early in the design process and then revisited during the design and build. The process of breaking a big system into a collection of subsystems (or layers/domains) is called decomposing.

There are certainly many ways to decompose a large system into layers. In each design, it is important to decide when the architecture has enough layers to do the job, but not so many as to add unnecessary complexity. A best practice is to add a layer and then apply the logical decomposition test. If it fits the requirements or, alternatively, performance is not what it needs to be, then the next step is to add another criteria. As a best practice, these are the three most commonly used metrics for evaluating the efficacy of a layer in a multi-layer architecture:

- Discovery timing
- Application of resources
- Latency

Security

Security is important throughout all the layers. With a DDS databus, security is a separate implementation that enables security without impacting the logic and function of the application. This means that security can evolve without having to reprogram applications, with the reverse also true.

The DDS databus enables security requirements by providing authentication, access control, cryptography and logging capabilities for secure data distribution. DDS security is independent of application logic and can, therefore, be applied separately without affecting the application's functionality. This separation enables the application data model, logic and security requirements to evolve separately with little to no impact to one another.

USE CASES

The following are sample industry-specific use cases. It is important to note that the inherent flexibility of the DDS databus means that the technology doesn't dictate or restrict how a layered databus architecture might be set up. The layered databus is not prescriptive; it's highly customizable. Note that customizable does not mean proprietary programming; it refers to the inherent flexibility within the standard to accomplish something in many different ways. Each of these use cases could be designed differently, while still adhering to—and getting all the benefits of—a standards-based approach.

Healthcare: patient monitoring

This example focuses on monitoring at the patient bedside. Typically, patient monitors are based on proprietary software and hardware that provide information on the patient's condition. Often these independently monitor specific conditions and do not interact with other equipment.

As shown in Figure 5, there are many other devices at the bedside, such as pumps, ventilators and EKGs. For the most part, these devices operate independently and are usually based on proprietary designs. This means they have their own interfaces that the clinicians have to read, and the data output from each device has to be recorded separately. In order to get a complete picture of the patient's health, the clinician has to consult numerous records from these different systems, which can lead to errors and delays in care. Some of this information is collected intermittently and some continuously. Some of this data is critical to the patient's status, while other data is of lesser importance.

Layer 1, as shown here, provides a communications platform at the bedside for all these independent devices to communicate with each other as needed.

Layer 2 allows clinicians to view any patient bedside data, selectively, from any location. This selectivity is enabled by filtering, which is just one of the capabilities enabled by RTI Routing Services.

Layer 3 provides a communications platform for all of the hospital enterprise-level databases. These databases need to collect information, such as radiology reports and images, without impeding the flow of information to and from clinicians and all of the data from bedside monitoring systems. This is another capability of RTI Routing Services and the layered databus design: the ability to isolate critical data from non-critical data.

Transportation: autonomous vehicles

One of the many benefits of the layered databus is the unified data model. Autonomous vehicles may seem like an entirely new type of transportation. But, in fact, many of their subsystems are legacy and proprietary, from the braking system to the CAN bus. In this example (Figure 6), Layer 1 is the databus for vehicle control, which is a mix of autonomous and non-autonomous elements. Layer 1 can be utilized as the communications platform between the lower bandwidth legacy systems and the higher bandwidth sensors required to provide autonomous capabilities.

Layer 2 may also be within the vehicle. In this example, Layer 2 supports telematics. Note that there is a gateway (RTI Routing Service) between the two layers. One of the important roles of the gateway is to isolate critical safety-related data flowing across Layer 1 from non-critical telematics in Layer 2. The design goal is that any failure in telematics will not affect the safe operation of the vehicle. The DDS databus includes built-in security to protect the system against unauthorized traffic.

Layer 3 provides the communications platform for the municipal cloud. These cloud-based services provide critical communications with Layer 2. This constant flow of two-way communications provides vital information on road conditions, weather patterns, accidents and detours, etc.

As the diagram shows, there can be other layers for vehicle-to-vehicle communications, fleet management communications and much more.

With the unified data model, engineering teams can use a consistent data model from the smallest sensor in the car up to the cloud. In contrast, proprietary implementations may have different teams working on different protocols and interfaces. The unified data model not only eliminates this resource drain, it also means that if there are changes to the platform, protocols or architecture, it will not mean redesigning applications.

Utilities: distributed energy resources

In recent years, distributed energy models, including microgrids, have been proliferating because of the rise of renewable energy. A microgrid operates in a defined area, e.g., a neighborhood, where the microgrid operator can manage the microgrid without affecting the integrity of the entire grid. As shown in Figure 7, the Layer 1 databus is the communications platform that provides automated discovery of any new renewable energy source, such as solar panels and wind turbines. This eliminates dependence on technicians to set up monitoring and control systems for each new system that joins the microgrid, which, in turn, keeps costs down.

Layer 2 adds a databus for analytics to support predictive analysis that can be used to optimize the performance of one or all of the microgrids. For example, this data can also be used to minimize an unexpected power draw from the main grid in case of a microgrid failure.

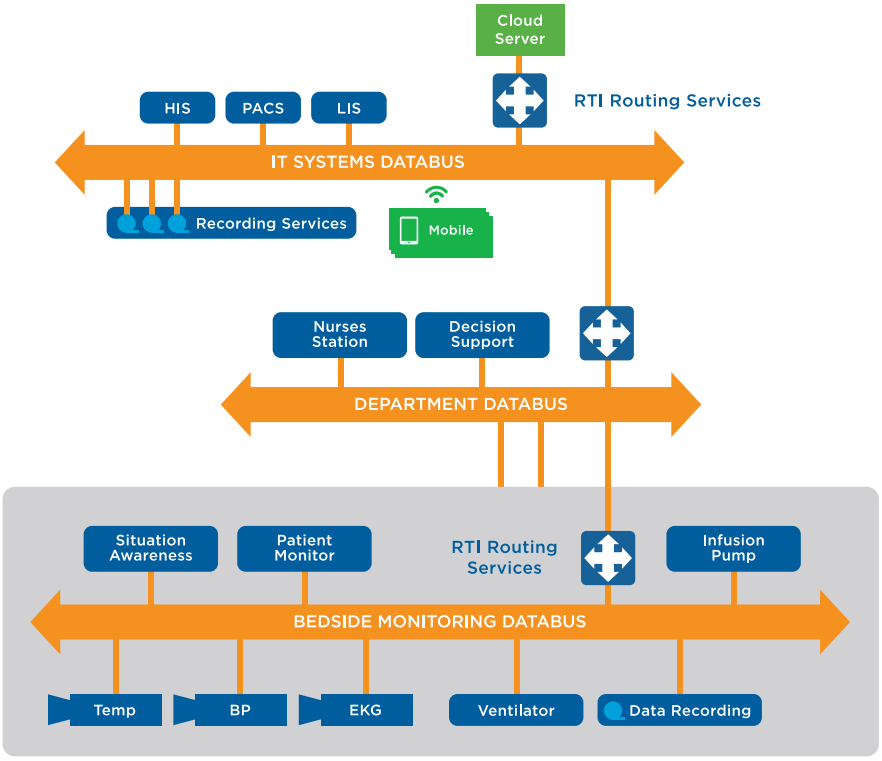


Figure 5: The layered databus architecture in a hospital use case.

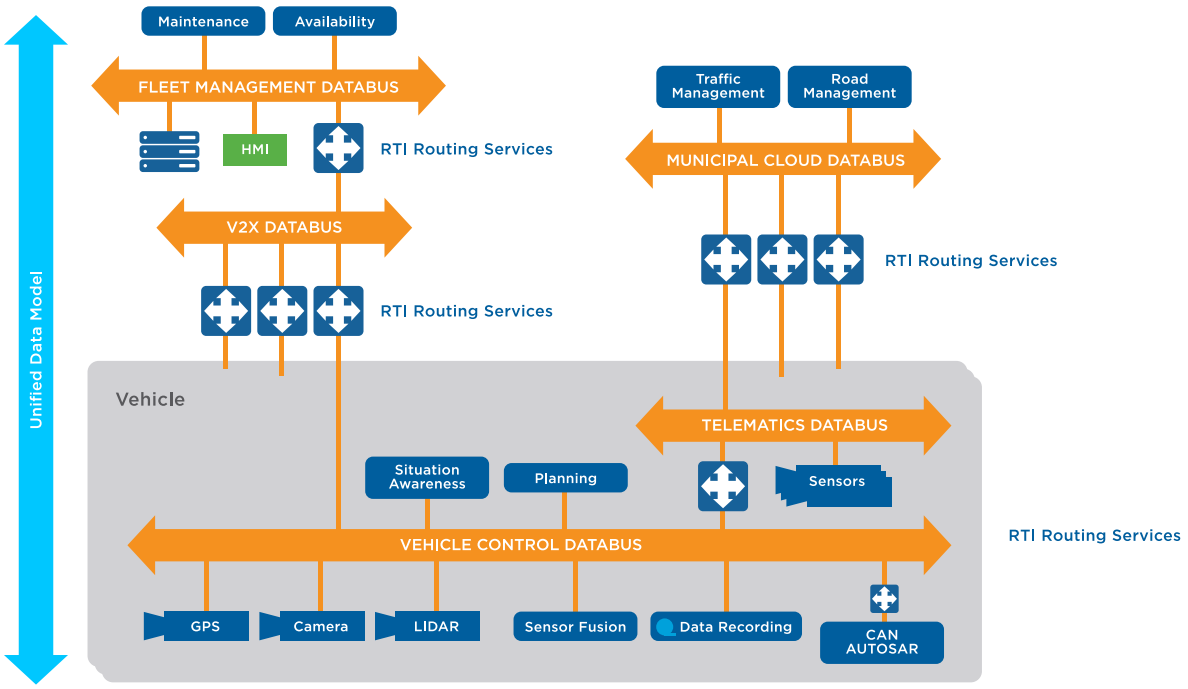


Figure 6: : Sample layered databus architecture used in an autonomous vehicle scenario.

Layer 3 talks to the main grid, where regulation is extremely heavy. Each authority served by Layer 3 has its own legacy processes. The microgrid has to be able to provide the necessary compliance information in whatever legacy format is required. This highlights one of the features of the DDS databus, which is the ability to provide interoperability with legacy protocols without any special programming.

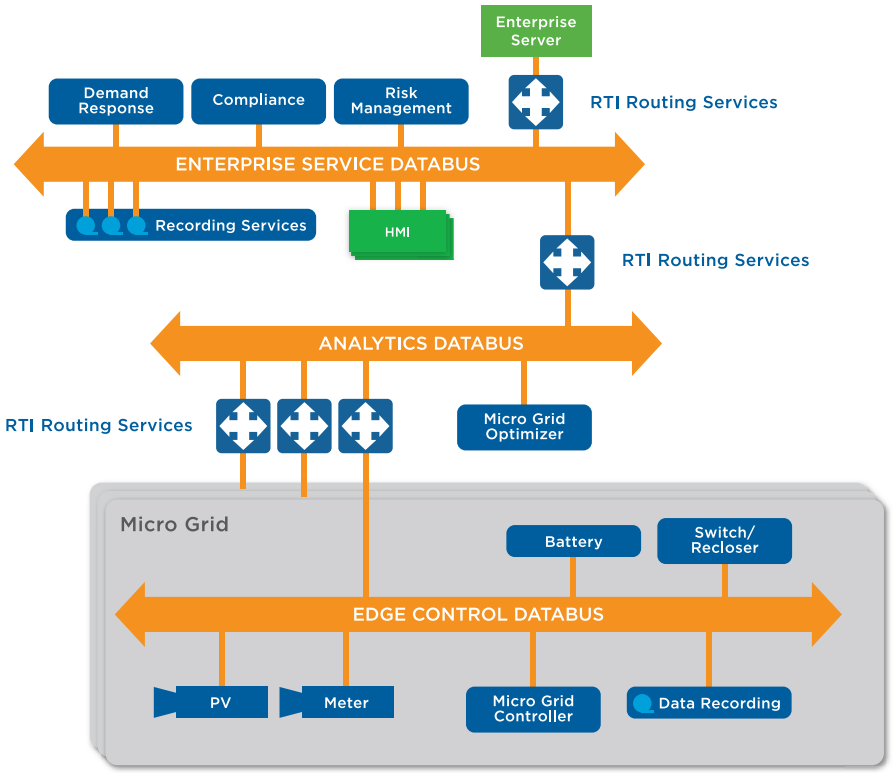


Figure 7: Sample layered databus architecture used in a distributed energy resource (DER) environment

CONCLUSION

A layered databus architecture creates a virtual plug-and-play communications platform to modernize and expand legacy applications and systems. It eliminates the need to recompile, rebuild, revalidate and reset proprietary monolithic systems when a new subsystem needs to connect.

Layered DDS databuses solve complex problems of expansion and scalability without sacrificing the granularity and control that complex, interconnected designs require. The layered databus architecture makes it comparatively easy to enable proprietary systems to interoperate with new systems. Layers are transparent and operate automatically: they separate and share based on the needs of the domain publishers and subscribers.

The layered databus architecture preserves the integrity of legacy systems, while opening up designs to a limitless array of new participants. Industrial engineers can utilize the layered databus to create a sustainable architecture that bridges from their existing brownfield systems to modern systems while allowing for future flexibility and requirements.

For more information about how a layered databus can work in your system architecture, please visit www.rti.com.

ABOUT RTI

Real-Time Innovations (RTI) is the largest software framework provider for smart machines and real-world systems. The company's RTI Connext® product enables intelligent architecture by sharing information in real time, making large applications work together as one.

With over 1,500 deployments, RTI software runs the largest power plants in North America, connects perception to control in vehicles, coordinates combat management on US Navy ships, drives a new generation of medical robotics, controls hyperloop and flying cars, and provides 24/7 medical intelligence for hospital patients and emergency victims.

RTI is the best in the world at connecting intelligent, distributed systems. These systems improve medical care, make our roads safer, improve energy use, and protect our freedom.

RTI is the leading vendor of products compliant with the Object Management Group® (OMG) Data Distribution Service™ (DDS) standard. RTI is privately held and headquartered in Sunnyvale, California with regional headquarters in Spain and Singapore.

Download a free 30-day trial of the latest, fully-functional Connext DDS software today: https://www.rti.com/downloads.

RTI, Real-Time Innovations and the phrase "Your systems. Working as one," are registered trademarks or trademarks of Real-Time Innovations, Inc. All other trademarks used in this document are the property of their respective owners.
©2020 RTI. All rights reserved. 50048 V0 0920

Your systems. Working as one.

CORPORATE HEADQUARTERS
232 E. Java Drive, Sunnyvale, CA 94089
Telephone: +1 (408) 990-7400
Fax: +1 (408) 990-7402
info@rti.com

rti.com
company/rti
rti_software
connextpodcast
rtisoftware
rti_software