WHITEPAPER

The Secret Sauce of Autonomous Cars

Stan Schneider, CEO, Real-Time Innovations, Inc.

Abstract

Intelligent vehicles are complex distributed systems. An autonomous car combines vision, radar, lidar, proximity sensors, GPS, mapping, navigation, planning, and control. These components must combine into a reliable, safe, secure system that can analyze complex environments in real time and react to chaotic environments. Autonomy is thus a supreme technical challenge. An autonomous car is more a robot on wheels than it is a car. Automotive vendors suddenly face a very new challenge.

Of course, the challenge isn't completely new; autonomy has a rich technology heritage in other industries, including aviation drones, space robotics, and underwater vehicles. Autonomous cars can leverage this base, but streets and highways are much faster, more chaotic environments. Human cargo raises the stakes for reliability and safety. Passenger vehicles are already very complex systems. Thus, cars take the autonomy challenge to a new level. They need industrial-class reliability, true real-time response, and powerful system integration.

So, what's the secret sauce? System components interact through data flow. Complex systems, therefore, operate by controlling that dataflow exactly as needed by each component. This technology is called data-centric connectivity. It today runs intelligent robots and reliable power and medical systems.

Data-centric connectivity was originally developed for autonomous systems. Unlike messaging technologies, it directly controls data interactions. It removes all of the complexity of managing data communications from components. It excels at highly reliable, complex system integration. It is fully standardized, proven in hundreds of industrial systems, and already controlling many autonomous planes, robots, submarines . . . and cars.

Data Centricity History in Autonomy

Data-centric communications originated in autonomous robotic systems. Researchers building autonomous air and space vehicles at the Stanford Aerospace Robotics Laboratory developed the core technology. In 2004, the technology was codified in a standard called the Data Distribution Service (DDS).

Since then, DDS expanded into thousands of applications in defense, medical, power, mining, communications, and transportation. Today, DDS is everywhere there is autonomy. It integrates the world's most advanced drones, NASA planetary rovers, unmanned military ground vehicles, and autonomous surface and underwater robots. RTI was the architectural lead for the US ground station design; DDS now forms the core connectivity for essentially all ground stations. An RTI-based nationwide system called Ground-Based Sense and Avoid (GBSAA) will soon allow drones in US National Air Space. And, as this paper outlines, DDS is a key component in the exciting emergence of autonomous cars and trucks.

Figure 1. Data Centricity in Autonomous Systems. RTI was founded by researchers in autonomous systems at Stanford University. RTI runs autonomous systems today in air, land, sea, and space. RTI middleware integrates most US unmanned air systems ground stations, runs dozens of NASA rovers, and integrates Advanced Driver Assistance Systems and full autonomy for Volkswagen.

In all of these systems, DDS provides reliable, responsive distributed control. In contrast to traditional messaging technologies, DDS focuses on real-time performance, high reliability, system integration, precise dataflow control, and complex software decoupling. It greatly accelerates and simplifies distributed system development. DDS is the only technology that can deliver microsecond latency, IEC 26262 safety certification, top security, and proven operation in billion-dollar product lines.

Autonomy Demands

Autonomy is a sliding scale. Most classification systems differentiate by the amount of time between human intervention. The most common classification in the automotive industry defines these “levels”:

- Level 1: Automated single control functions, such as speed control, emergency braking or lane keeping.
- Level 2: More than one control function is automated. The driver is expected to be available for control on short notice.
- Level 3: The vehicle takes control most of the time. The driver is only needed for occasional control with comfortable transition times.
- Level 4: All functions are automated. The vehicle can operate without a human driver.

There is a world of difference between these levels. Level 1 systems require almost no intelligence. “Smart” cruise control, for instance, is doable with a simple feedback loop wrapped around a single sensor. Level 2 systems are often called Advanced Driver Assistance System, or ADAS. This is much harder than level 1, but still does not require sophisticated sensing, situational awareness or planning. These are by far the greatest challenges. Level 3 requires most of the sophistication of full autonomy. It falls back on the human driver in tough situations.

Level 4 is the current golden target of automotive systems. It is worth pointing out that autonomous air vehicles have been operating at level 4 for many years. Drones and even passenger-jet flight management systems are capable of flying entire missions or routes with no human intervention. Of course, the environment is much simpler; there’s not much to hit in flight.

That highlights a key point: environmental complexity is key. Autonomous cars can operate in very chaotic urban environments or very structured freeways. Structured environments make sensing and situational analysis much easier. Much of the complexity is independent of speed. Parking lots, for instance, are rife with unexpected obstacles, pedestrians, and sensing challenges. Autonomy will likely be practical in structured freeways, even at high speed, before it will work reliably in general driving.

The Importance of Data Centricity

Systems are all about the data. Distributed systems must share and manage that data across many processors and applications. The strategy to understand and manage this state is a fundamental design decision. Fundamentally, data centricity is a technological approach and philosophy for how to manage state.

What is Data Centricity?

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, and data object create, read, update, delete (CRUD) interfaces.

Real-time distributed systems need data-centric communications. However, 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, a very special file. A database has known structure and access control. A database defines "truth" for the system; as long as data in the database if not corrupted or lost.

Like a database, data-centric middleware imposes known structure on the transmitted data. DDS 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.

DDS doesn't (usually) store any data, but it does manage moving information in the same data-centric way that databases manage stored information. Because of that, DDS is often called a "databus."

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

Data-Centric Middleware is Different

Because it is data centric, DDS is not like other middleware. With knowledge of the structure and demands on data, the infrastructure can do things like filter information, selecting when or even if to do updates. It directly addresses real-time systems. It features extensive fine control of real-time "Quality of Service" (QoS) parameters, including reliability, bandwidth control, delivery deadlines, liveliness status, resource limits, and security. It explicitly manages the communications "data model," or types used to communicate between endpoints.

At its core, DDS implements a connectionless data model with the ability to communicate data with the desired QoS. A DDS-based system has no hard-coded interactions between applications. The databus automatically discovers and connects publishing and subscribing applications. No configuration changes are required to add a new application to the network. The databus 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.

How Does DDS Work?

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 databus, 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.

The databus ties all the components together with strictly-controlled data sharing. The infrastructure implements full quality of service (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.

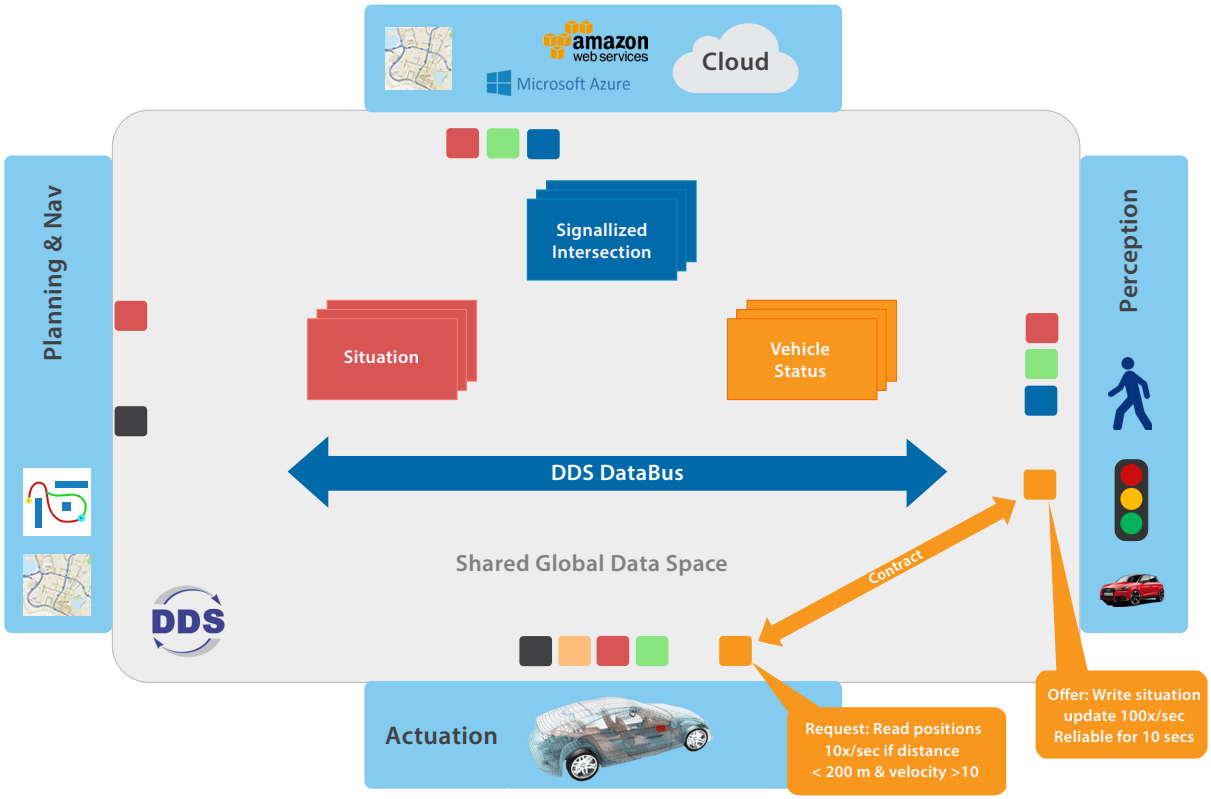

Figure 2. Data-Centric Communications. The databus links any language, device or transport. It automatically discovers information sources, understands data types, and communicates them to interested participants. It scales across millions of data paths, enforces millisecond timing, ensures reliability, supports redundancy, and selectively filters information. Each path can be unicast, multicast, open data, or fully secure. In the figure, a perception system tracking many objects will send only those close to, and approaching, the vehicle. Although the perception system can update at 100Hz, the receiver only needs it at 10Hz.

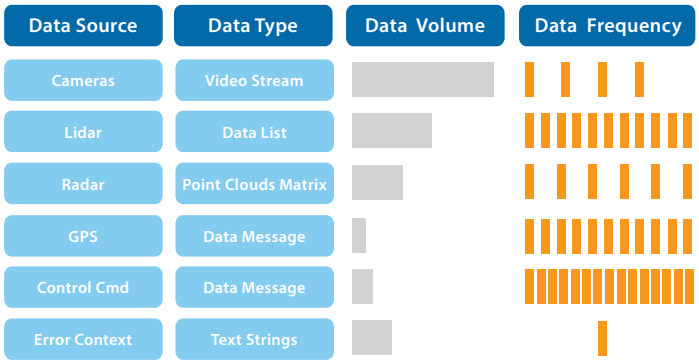

Figure 3. Complex Data Flow. A complex system must support many data types and sources. Some are very high volume; others are fast. QoS control permits each module to specify every data relationship required for operation. The specifications are much more than just what data is needed. They include QoS like update rates, reliability, data availability guarantees and more. Infrastructure that specifically sends exactly the right information to exactly the right places at the right time makes system development much easier.

The databus 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 for reliable, high-performance, large-scale IIoT systems. As of this writing, there are at least 13 implementations of DDS. Several are backed by commercial vendors. RTI provides the leading implementation.

How Does DDS Help Autonomy Development?

DDS was developed for autonomy. It has very unique properties that map well to autonomy. Let's take a look at some of those properties by examining actual applications.

Ensure Reliable Data Availability

Autonomous systems cannot stop, even for a few microseconds. RTI has experience with many reliable systems.

DDS provides this availability by making it easy to implement a fully-redundant architecture, including computing, sensors, networking, storage, and software. Since it communicates state instead of sending messages, DDS systems naturally support multiple producers and consumers of every dataflow. RTI also allows redundant network transports by being intelligent about delivering data only once upon receipt.

Thus, adding redundancy is simple.

Figure 4. 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.

Guarantee Real-Time Response

There are many of ways to characterize “real time.” All systems should be “fast.” To be useful, we must specifically understand which timing requirements drive success.

“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. RTI Connext DDS achieves real-time performance by sending information directly between peers with no servers or brokers, by using multicast when possible, and by optimizing both send and receive code paths.

Figure 5. Real-Time Applications. RTI middleware is used in many systems that require controlled latency. Examples include navy weapons systems that must track and shoot down incoming missiles, medical robotics with 3 kHz sample loops, computed tomography imagers with extremely fast X-ray and servo control, and even multichannel real-time video switches.

Ease System Integration

Intelligent systems manage many variables and very complex-state relationships. DDS excels at decoupling the complexity.

Large systems contain many “modules,” each typically an independently-developed application built by an independent team of developers. Module scale quickly becomes a key architectural driver. The reason: 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.

Figure 6. Challenging System Integration. Modern cars, like many intelligent systems, must integrate the work of many teams. The DDG-1000 ship, for instance, integrates thousands of software modules built by 1500 teams of programmers. Audi’s hardware-in-the-loop simulation system must support the development of hundreds of ECUs. Hospital systems integrate hundreds of types of devices. Data-centric technologies directly manage the interfaces between these teams, greatly easing system integration.

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 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. Data-centric systems expose and control interfaces directly, thus easing system integration. 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. However, databases provide 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, then adapt to deliver to each endpoint in the schema what the endpoint expects. These systems thus decouple application interface dependencies, allowing large projects to evolve interfaces and make parallel progress on multiple fronts.

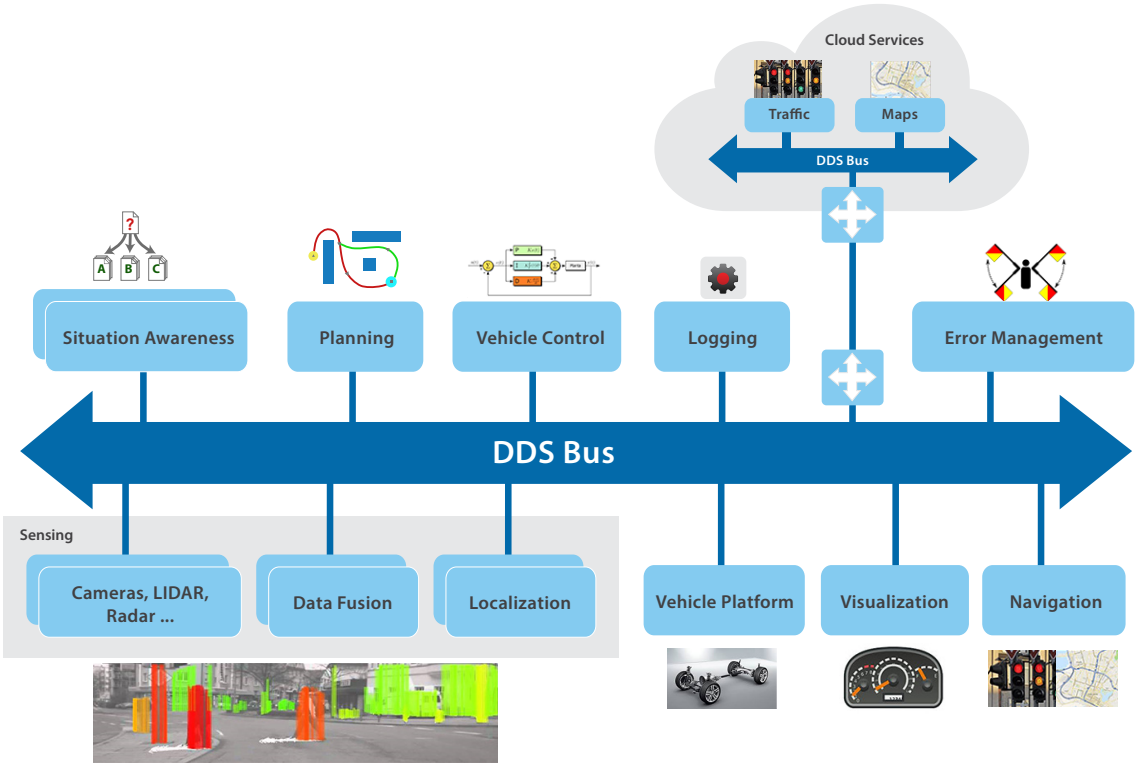

Figure 7. Autonomous Car System Architecture. DDS integrates all the components in a typical autonomous car design. The data-centric interface means that all module interactions are strictly controlled for schema matching, rates, reliability, and system health. Each of these components is a complex module on its own. Powerful system integration support lets teams work independently with the assurance that the infrastructure will directly support the data interaction needed for system-wide operation.

Build Security in From the Start

Security is 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-n-play discovery, so must security. Security that is this intimately married to the architecture can be imposed at any time without changing the code. Security becomes just another controlled quality of service, albeit more complexly configured. This is a very powerful concept.

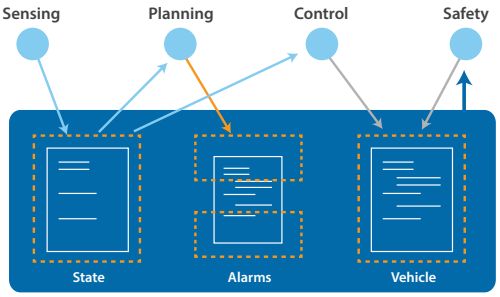

Figure 8. Dataflow-Level Security. The DDS standard security model supports all the key security components: authentication, access control, encryption, tagging and logging, and even non-repudiation. The secured elements are not "users" or "pipes," but actual system modules. Simple configuration controls which modules can read and write which data items. Thus, the security architecture exactly matches the connectivity architecture. In the example figure, the sensing module can only affect state. Planning can read state and set alarms. Control can read state and affect the vehicle. Safety can read anything and control the vehicle. This level of security limits the impact of a breach of any one module.

Make Deployment Flexible

Data centric design implies that only data interactions matter. Thus, modules may run anywhere in the system. There can even be redundant copies of modules running for reliability. Location, chip architecture, language, and operating system differences are completely transparent.

Ease Safety Certification

DDS is already running many safety-critical systems. One of the most demanding is a system called Ground Based Sense and Avoid (GBSAA), which allows autonomous planes to fly in the US National Air Space (NAS). As part of getting this system running, RTI put a version of its Connext® DDS product through full DO178C-Level A Certification, the FAA's most stringent standard. Certification evidence includes 940 High-Level Requirements, 3,680 Low-Level Requirements, 3,400 test files (many of which have multiple test cases), and 99.88% code coverage testing.

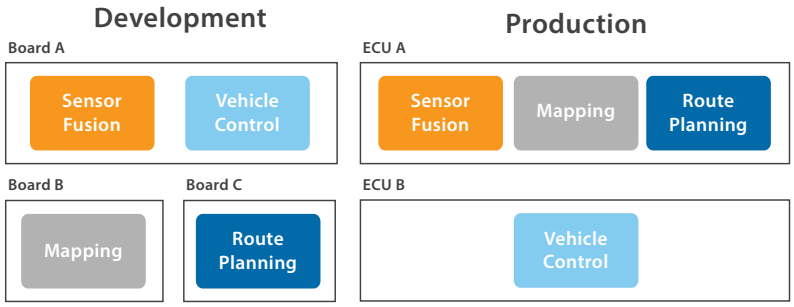

Figure 9. Deployment Flexibility. Data centricity separates physical location, transport, chip architecture, language, and operating system from operation. Thus, system designers can move modules between physical locations. As long as the data relationships are satisfied, the system will work. Configurations can change as teams move through evaluation, prototyping, testing, production and maintenance.

DO178C-Level A is more demanding than the most important standard affecting autonomous cars, the ISO 26262 road-vehicle functional safety specification. Thus, RTI's Connext DDS Cert product can directly support vehicles that require safety certification. Importantly, teams can develop with the very functional "full" DDS, and then deploy critical modules with fully certifiable middleware.

Certified middleware also separates user code modules. As an isolating layer, certified middleware divides the system into provably separate modules. Thus, the system can easily combine critical components with non-critical subsystems. RTI's decentralized support for redundancy, automatic failover, server-less operation, and dynamic upgrades makes expensive certified systems much more modular and therefore much less expensive to certify. Because the much-smaller certified version interoperates with the "full" version, developers can mix-and-match functionality without increasing certification cost.

Summary

An autonomous vehicle is a robot on wheels. Connectivity is key to success for complex robotic systems. The DDS standard delivers reliable, flexible, real-time, secure connectivity proven in operational autonomous applications and billion-dollar product lines.

With extensive QoS filtering and delivery control, DDS implements an extremely optimized system. It adapts to all the varying components, including perception, mapping and navigation, connection of vehicle, decision and planning, display and visualization, and vehicle control. It can handle high-volume video and sensors as easily as low-latency feedback. It manages redundancy naturally; developers can implement double or triple redundancy with no extra work. It adds security control to all dataflows. It also provably separates modules, making certification easier and much less expensive.

Thus, DDS delivers microsecond latency, easy scalability, IEC 26262 safety certification and top security. Because it separates and controls module interfaces directly, DDS eases development, integration, debugging, and deployment.

In cooking, the sauce is critical; ties all the components into a working whole. Every chef knows: the right sauce makes the dish.

Data is the food that feeds distributed systems. Systems need the right sauce to tie components together. The right middleware – industry-proven, data-centric middleware – is the secret sauce that makes autonomous cars effective, safe, and practical.

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