Accelerating Time-to-Market

Using an Integrated High Assurance Software Stack

2022-07-27

Authors:

Paul Pazandak
Real-Time Innovations, Inc.
paul@rti.com

Alexander Lehmann
Hensoldt Cyber
alexander.lehmann@hensoldt.net

Nicholas Evancich
Trusted Science and Technology
nick@trustedst.com

Fabrizio Bertocci
Real-Time Innovations, Inc.
fab@rti.com

Sebastian Eckl
Hensoldt Cyber
sebastian.eckl@hensoldt.net

CONTENTS

FIGURES

1 INTRODUCTION

The cost to design, build, verify, and certify high assurance systems will rapidly exceed time and budget targets. This can add thousands or tens of thousands of hours of additional effort; and, for the highest levels of certification, exceed $100 per line of code. (CITE) These costs can be mitigated by building upon a high assurance software stack foundation that is already verified/certified. As a result, the software certification needed will be limited to the application code that you develop.

With support from DARPA, Real-Time Innovations (RTI) has been working in the area of formally-verified microkernels and software certification frameworks for the last six years. Our focus has been on exploring software architectures and technologies that will accelerate time to market for high assurance systems. Our software stack is built upon open-source technology based upon the formally-proven seL4 microkernel, and a data-centric communications technology using Object Management Group's Data Distribution Service (OMG DDS). The working implementations we will discuss are using Hensoldt Cyber's TRENTOS (a commercial operating system implementation on top of seL4), and RTI Connext Cert (a certifiable commercial software framework that is OMG DDS-compliant).

We walk through multiple software architectures that utilize this software stack, and present performance results from our reference implementations.

The purpose of this document is to present the details, use cases, and discusses performance of a high assurance software stack to expose readers to issues and challenges before starting their own efforts. We share a cost-effective approach to accelerate the design and implementation of their systems by building upon a high assurance stack (seL4 and OMG DDS). This paper provides the rationale, explains a software stack and outlines multiple software architectures that use it. This should be useful to IOT architects, developers, integrators, and safety/security assurance personnel.

2 MOTIVATION

The process of building trustworthy and high assurance systems is complex, costly, and requires significant expertise. The end goal is to create a complete software-hardware solution whose components, both individually and collectively, meet your customers' required levels of assurance for safety and security. This will vary depending upon the standards that are required. For example, RTCA DO-178C[1] for flight safety airworthiness, and ISO 26262 for autonomous vehicles. Within each of these, there are multiple levels of certification corresponding to the

[1] See standards, certification companies, and software references at the end of the paper.

level of criticality (the role) that the component has. For example, within DO-178C there are five levels:

- Level A: Catastrophic: prevents continued safe flight or landing, many fatal injuries
- Level B: Hazardous/Severe: potentially fatal injuries to a small number of occupants
- Level C: Major: impairs crew efficiency, discomfort, or possible injuries to occupants
- Level D: Minor: reduced aircraft safety margins, but well within crew capabilities
- Level E: No Effect: does not affect the safety of the aircraft at all

Given that the cost of software certification alone can be exceedingly expensive, depending upon the certification level required, there is a strong motivation to find and use as many pre-certifiable components as you can in order to significantly reduce the program risk and cost and time to delivery.

While you must develop your application code, you can (and should) avoid developing as much of the code that it sits upon. This includes the components you will need for on- and offboard communications, along with the operating system. It is important to highlight that your choices early on can have a significant impact on your overall certification costs. For example, choosing the right programming language with an ecosystem of certified tools (such as Ada), may be worth the investment. We suggest working with certification experts as early in the software development process as possible.

In this article, we propose a verified stack to accelerate safety/security accreditation for consideration. It involves the combination of seL4 and the Object Management Group Data Distribution Service (DDS). seL4 is a mathematically formally verified microkernel[2] that has been long-funded and supported by DARPA; and DDS is an open standards-based communications middleware.

The use of seL4 and DDS can significantly reduce the time a customer needs to invest to develop and commercialize their system. That is, they will only need to implement their own application functionality and certify their own code - not the rest of the stack. This will accelerate time to market significantly.

Leveraging both IRAD and DARPA funding, we have created a high assurance software stack that will significantly reduce time to market by:

1) Providing a formally verified software stack that is ready for safety certification as a starting point
2) minimizing the application code size, reducing what users need to develop, certify, and maintain

[2] *https://cseweb.ucsd.edu/~dstefan/cse227-spring20/papers/sel4.pdf*

Utilizing the OMG DDS open standard enables the ability to rapidly assemble loosely coupled (distributed) software components into a working system

As a part of this project, we have built several reference architectures that demonstrate the utility of this approach.

3 BUILDING TRUSTWORTHY SYSTEMS

General purpose computing, operating systems, inherent language features (e.g., C memory allocation), and software quality issues have led to a lack of inherent security and resiliency in systems throughout industry. This has resulted in many security breaches that have had dire consequences to national security. It is necessary to design assured systems based on appropriate techniques and tools by applying sound security and engineering principles.

Generally speaking, building an assured system entails a thorough understanding of the problem domain, deep analysis of domain-specific workflows and requirements, careful architectural considerations and design trade-offs, vetted development, proper configuration and managed deployment of the final product. This level of care will also be needed throughout the product lifecycle. Specifically related to system architecture, leveraging hardware and software techniques and tools for enhanced security boils down to applying sound security principles to suitable targets such as memory access (e.g., the Principles of Open Design, Least Privilege, Separation of Privilege, and Complete Mediation). Other research and development efforts may adopt different applications of such principles to their particular environments and design goals.

3.1 CHALLENGES

One common theme in developing a trustworthy system architecture is related to secure communications, which include communications of the system with external parties and those among internal entities of the system itself. It is relatively straightforward for a developer to adopt the Complete Mediation principle and check integrity and confidentiality using some cryptographic methods, especially for external communications. For internal communications, some kind of broadcast via a bus interface (e.g., MIL-STD-1553) is usually adopted due to reasons such as legacy support and easy integration. These practices are not sufficient to provide the needed security, resiliency, assurance or even efficiency.

First, while the cryptographic algorithms are standardized, correct application of cryptography, particularly key management, remains a major challenge in implementation. Example vulnerabilities related to inappropriate use of cryptography abound. Moreover, broadcast-based bus interfaces are well known to be vulnerable to myriad attacks, since by nature the bus is an “open-party-line” that each module attached to the bus can listen to, receive and send messages. A well vetted and standardized way to apply cryptography is needed, and security should be applied to bus-based communications.

Second, in current design and development practices, even when cryptography is correctly applied, the message-based communication model still poses significant technical challenges in that each application/module will have to understand data, information and context separately, usually after receiving a sequence of messages. All of the knowledge of what is flowing over the network is opaque in message-based communications. So, while this model makes sense in the traditional context of protocol development, it does not suit the needs of building assured systems where mission and application contexts are ubiquitous. In other words, it is not the raw "message" that matters; it is the data and information in context that ultimately matter. That is, we need to know what data is moving over the network. Therefore, a data-centric approach (as opposed to message-centric) is more desirable because it provides critically needed network packet transparency. This approach needs to be real-time, secure, and efficient.

Building high assurance systems will require deep expertise, a lot of patience, and substantial funding. Multiply that by a factor of three if your system needs to be certified to some standard: you will need in-house expertise in software certification along with an outside certification partner; and, the path to market will be on the order of years. In our experience, you should expect certification costs alone to range from $5-$300 per single line of code (SLOC) depending upon the level of certification needed.

3.2 SOLUTIONS

For the reasons just enumerated, it is simply too costly in terms of funding and time to build a high assurance system from top to bottom. On the contrary, *the goal should be to develop as little code as possible*. The more *proven/certifiable* code that one can acquire or license, the less one will need to design, develop, maintain, and certify. This will expedite development efforts and significantly lower costs. A high assurance software stack provides this (as depicted in the Figure 3-1. High assurance stack.).

Figure 3-1. High assurance stack.

In this paper we do not review, compare, or debate alternative stacks. Rather, we delve into a high assurance software stack that was developed in a DARPA funded effort over the last five years by RTI Research. We have designed and implemented architectures for secure medical systems and defense applications. It is suitable for both embedded systems with tight resources and more capable hardware.

The role of this stack is to provide a proven foundation. It is composed of a real-time operating system (RTOS) that has been verified or certified (a *safety* RTOS), and a distributed communications middleware. In our stack, for the Safety RTOS we chose the open source seL4

separation kernel (sel4.systems). It is a mathematically provably correct microkernel that will provide both time and space separation between running processes. It guarantees that there will be no unintended data leakage between processes, and that one process cannot impact the operation of another. This provides greater system resilience and security (these are also attributes of a multiple independent levels of security (MILS) solution[3]). On top of this, we use a middleware framework based upon the Object Management Group Data Distribution Service standard (OMG DDS). The subsections below describe the motivations for each.

3.2.1 A MicroKernel Foundation

To understand the need for a secure microkernel, like seL4, it is helpful to start with a closer look at kernel design principles in general. As shown in Figure 3-2, there are two main kernel design approaches – the monolithic kernel and the microkernel. In the former one, all code required for providing typical OS services is directly implemented in the kernel itself. The kernel executes in the privileged mode of the hardware, meaning that all code is granted unrestricted access and control of all system resources. This type of implementation might be beneficial to the overall system performance, but it can lead to dangerous situations if any of the kernel components feature some type of malfunction – a state that could be exploited by an attacker. A prominent example is provided by the Linux kernel, which – containing more than 20 million lines of code – can be expected to contain a certain number of bugs providing potential attack channels.

In contrast, the microkernel design copes with this drawback by drastically reducing the trusted computing base (TCB), meaning the subset of code in the overall system that must be trusted to operate correctly. A microkernel follows the design principle of having the kernel contain only the most fundamental mechanisms (e.g. IPC, scheduling). All remaining OS functionality must be transferred to the unprivileged user mode, thereby running encapsulated within isolated sandboxes. This approach protects the kernel processes from any interference from the outside, only allowing communication that is explicitly wanted. For a well-designed microkernel like seL4 this means that code base can be reduced to the order of ten thousand lines of code. This drastically shrinks the attack surface. The general performance problem of typical microkernels, induced by the significant communication overhead due to the additional kernel entries/exits and context switches, was solved back in the mid-‘90s by Jochen Liedtke[4]. He accelerated the underlying IPC concept and designed the first L4 microkernel.

[3] *https://www.iiconsortium.org/pdf/MILS_Architectural_Approach_Whitepaper.pdf*

[4] *http://www.cs.fsu.edu/~awang/courses/cop5611_s2004/microkernel.pdf*

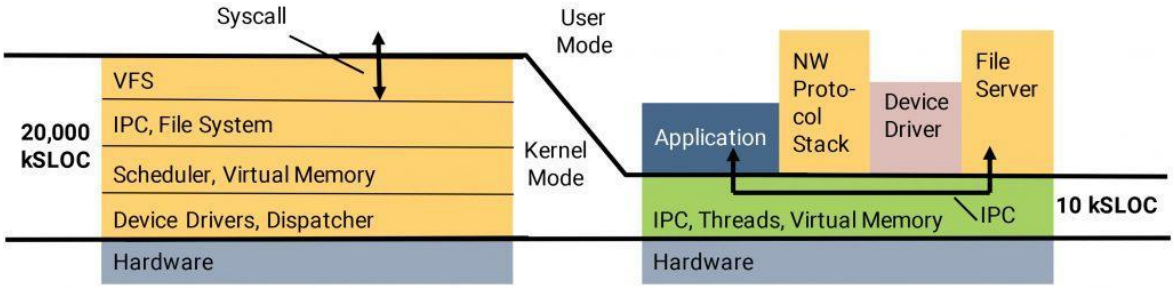

Figure 3-2. Kernel design approaches: Monolithic and microkernel (taken from seL4 white paper[5]).

3.2.1.1 THE SEL4 MICROKERNEL

seL4 joined the family of L4 microkernels in 2009, when it was first released to the public. Since 2014, it was made available as open-source software; and, as of 2020 it is officially maintained by the seL4 Foundation. In contrast to previous L4 designs, seL4 was implemented completely from scratch. The developers therefore were able to free themselves from this legacy (e.g. API and code), and yet benefit from all the positive and negative experiences of almost 20 years of L4 based kernel design. One of seL4's outstanding features is provided by its inherent and unique focus on the development of highly secure systems without compromising performance. Therefore, two main security design concepts were introduced: the utilization of *capabilities*, and the application of formal verification techniques. Within the context of seL4, capabilities represent a form of access token, which allows for a very fine-grained control over system resources. This supports the kernel's isolation properties. Additionally, with the help of formal verification techniques, the absence of bugs inside the kernel implementation has been mathematically proven with respect to its specification. This is an achievement that still leaves the seL4 kernel as the world's most advanced and most highly assured OS kernel.

3.2.1.2 CAMKES – AN ABSTRACTION LAYER ON TOP OF SEL4

Nevertheless, the actual implementation of services directly on top of the seL4 microkernel API itself can be quite a challenging task. To provide easier access to the underlying concepts, various helper libraries were developed, trying to hide the low-level kernel mechanisms of seL4. This makes the microkernel's security features and its mechanisms (e.g. IPC, memory management) easier to use for non-experts. The CAmkES (Component Architecture for microkernel-based Embedded Systems) framework extends this approach by providing "a software development and runtime framework for quickly and reliably building microkernel-based multi-OS systems."[6]

As a result, a layered component architecture for the separation of concerns was established, allowing for model-driven development. This enables the ability to auto-generate CAmkES configuration files from system models, which will reduce configuration complexity.

[5] The seL4 Microkernel, *https://cdn.hackaday.io/files/1713937332878112/seL4-whitepaper.pdf*

[6] Cited from: *https://docs.sel4.systems/projects/camkes*

CAmkES consists of a clear set of building blocks. Its primary purpose is to support the development of statically configured embedded systems. Therefore, CAmkES provides a dedicated language used to describe the components and their respective interfaces, as well as the composition of complete component-based systems. It also offers tooling support for processing these descriptions and automatically translating them into C code. In addition, respective proofs are generated. The resulting glue code is combined with programmer-provided component code, leading to the creation of a complete and bootable system image. The overall setup is then neatly integrated into the existing seL4 environment and build system.

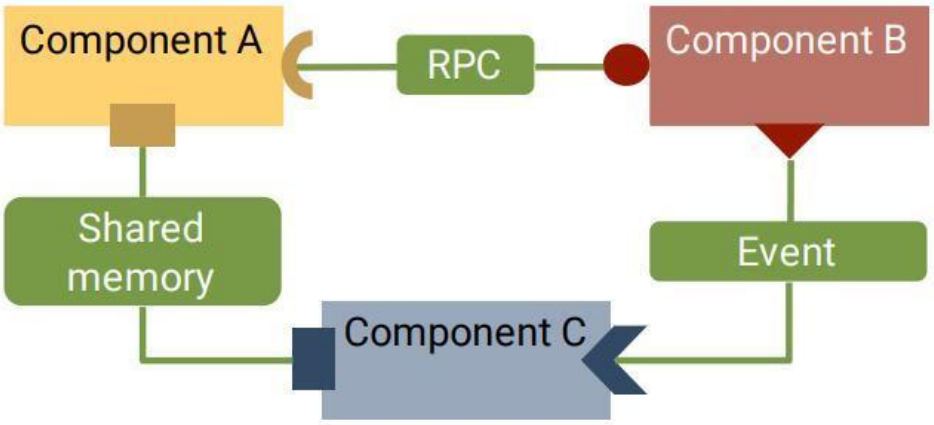

Figure 3-3. A Typical CAmkES system architecture (taken from seL4 white paper).

As can be seen in Figure 3-3, a typical CAmkES system consists of *components* (e.g. Component A) and their respective *connections* (e.g. RPC). When grouping all components and connections together, a *composition* is formed. If combined with a respective *configuration* section, a complete system (also called *assembl**y***) can be provided. The components hereby reflect seL4's isolation feature and comprise (at least) one thread for execution, an associated address space and required storage for associated capabilities. To enable interaction between components, CAmkES provides three basic connection types. Internally, they are mapped to a respective seL4 communication mechanism:

1. *remote procedure calls (RPCs):* synchronous communication, reusing seL4's IPC mechanism
2. *events:* asynchronous communication, reusing seL4's notification mechanism
3. *dataports:* bidirectional communication, suitable for exchanging large data between components via shared memory (size is constrained by available physical memory).

CAmkES and seL4 represent the latest technology advancements in secure microkernels and together comprise the "safety RTOS" shown in Figure 3-1.

Although seL4 and CAmkES provide rudimentary communications paths between processes that can be used for both on and offboard communications, all application-level code required to

enable this is left to the system developer. Consequently, this user space code will need to go through certification. However, significant expense in both time and money can be avoided by utilizing an existing certifiable COTS software communications framework, discussed next.

3.2.2 A Secure Communications Software Framework

The purpose of seL4 is to provide a reliable, safe, and secure foundation for applications that require it. This includes, for example: military systems, medical devices, robotics, autonomous vehicles, and energy systems. Without exception, these high assurance applications require a reliable and robust distributed communications capability, which is not provided by seL4.

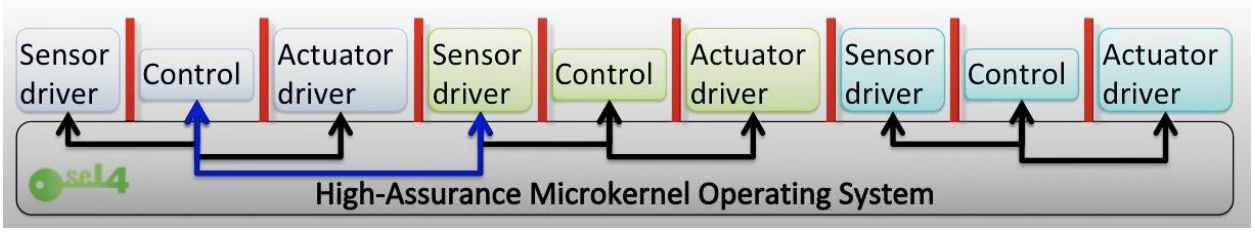

Figure 3-4. Microkernels provide process separation.

The seL4 kernel currently has a limited infrastructure for developing complex, high-assurance distributed systems. Since the inherent design of seL4-based architectures is to partition the application space, as shown in concept above (Figure 3-4), applications themselves need to deal with developing and managing *all* of the communication complexity. For example, configuring the inter-process communications (IPC) channels is complex. Developers will also need to define a protocol , serialization and deserialization, and several communications management features. Without standards, this will lead to numerous one-off approaches and as a result significantly limit component reuse within seL4.

The Object Management Group (OMG) Data Distribution Service for Real-Time Systems (DDS) is a real-time, secure, loosely-coupled, publish/subscribe software connectivity framework for distributed systems and is ideally suited as the communications layer for high assurance systems, including for any safety RTOS such as seL4. While there are other open source and commercial off-the-shelf communications framework technologies, those frameworks lack high assurance certification and at best they provide rudimentary all-or-none security.

For DDS, seL4 creates an enriched, lower cost, smaller footprint, high assurance foundation . For seL4, DDS provides an open standards-based communications protocol. DDS will significantly simplify seL4 inter-component/application development, reduce associated costs, and promote component interoperability in the seL4 development community. DDS is a solution that will standardize data distribution in a more consistent, secure and efficient manner. It provides a publish subscribe model that enables easier, faster and more secure distributed system development. Application developers can be alleviated from the burden of creating their own piecemeal, perhaps proprietary, and one-off solutions for message-based communications and

deciphering the message sequence, so that they can focus on domain-specific components and rely on DDS to provide standardized, secure interaction with other (local and remote) entities in the system.

DDS will significantly reduce the barriers of entry for companies and developers that decide to use seL4/CAmkES because it provides an abstraction layer that hides most of the complexity associated with developing applications on top of seL4. DDS will significantly reduce the development time and the need for seL4 subject matter expertise in-house. The next section delves further into what DDS is, and what features it provides.

3.2.2.1 OMG DDS

Most communications solutions are *message centric* and send their data over the network as encoded bits (opaque payloads) that the network cannot understand. All of the knowledge about encoding and decoding the data rests with the applications themselves. This forces every application to implement numerous features to support reliability, security, fault tolerance, scalability, and end to end interoperability.

In contrast, DDS is *data centric* (data aware) – meaning that both the data and data structures are accessible within DDS. This enables DDS to automatically handle encoding, security, optimized/reliable delivery, and more. Using DDS, developers define open data models that describe the structure of the data that will move between applications.

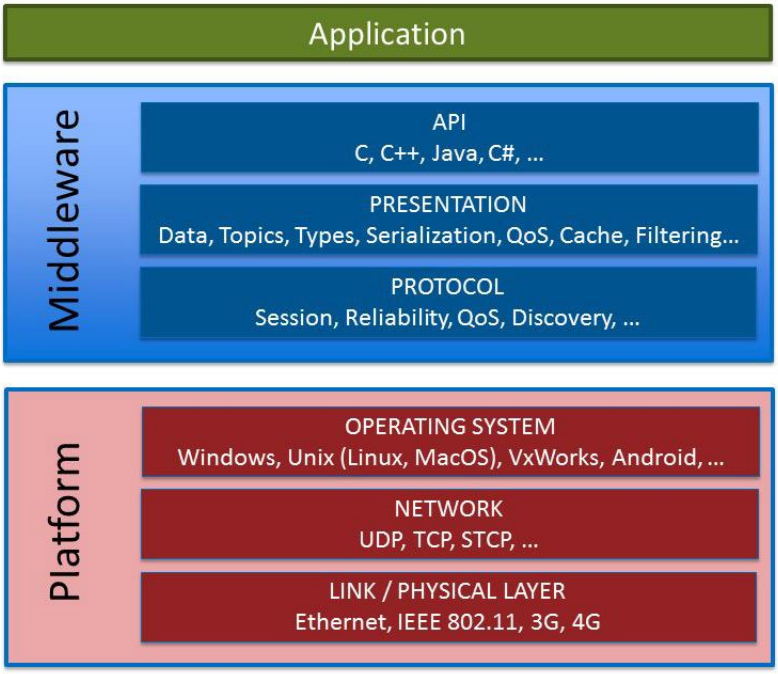

Figure 3-5. The DDS middleware operates between the platform and the application.

DDS is a software layer that abstracts the details of the operating system, network transport, and low-level data formats, hiding the details from the application. The same concepts and APIs are

provided in different programming languages allowing applications to exchange information across operating systems, languages, and processor architectures. Low-level details like data wire format, discovery, connections, reliability, protocols, transport selection, QoS, security, etc. are managed by the middleware.

Instead of creating brittle, point-to-point network dependencies, DDS communicates over the concept of *topics*. Applications simply declare what kind of data or *topics* (each topic has a well-defined data structure) they are interested in, and DDS delivers it. This eliminates the brittleness of requiring applications to identify specific endpoints that they need to talk to – DDS handles all of this, so developers can focus on application code and not on how to send data over the network.

DDS Security is unique because it provides Network Layer 4 fine-grained read/write access control to the data. This is in contrast to Layer 2/3 all-or-none options, including IPSec, MACSec, and D/TLS; these coarse-grained security alternatives expose more opportunities for exploitation. Using DDS fine-grained access control guarantees that only authorized applications can send and receive specific types of data over dedicated logical network partitions – enabling highly customizable communications supporting multiple security domains. This is *software-defined security* which is controlled through signed configuration files associated with each application.

DDS is loosely coupled and by using the concept of topics to communicate, it supports *location-independent processing.* This promotes system modularity and resilience, which are key requirements for modular open system architecture (MOSA) systems. DDS is also platform agnostic. This enables transparent interoperability between DDS applications independent of programming language, hardware, and operating system.

4 A FOUNDATION FOR BUILDING TRUSTWORTHY SYSTEMS

In the previous section we presented a description of a high assurance software stack built on seL4 and DDS, and we explained several of its benefits when developing critical systems. We made the case about why it provides a compelling foundation. In the section, we discuss a reference implementation of this stack that we assembled using implementations of CAmkES from Hensoldt (TRENTOS®) and DDS from RTI (Connext®), respectively. TRENTOS is the only production ready implementation of CAmkES. Connext is the only DDS implementation certified to the highest levels for both flight safety and automotive systems. We then present multiple example architectures and discuss design trade-offs.

We start by presenting a brief description of Hensoldt's commercial implementation of CAmkES, followed by RTI's commercial implementation of DDS.

4.1 THE FOUNDATION PART 1: TRENTOS – THE FRAMEWORK FOR A SECURE OS

Developed by Hensoldt Cyber, the Trusted Entity Operating System (TRENTOS®) is a novel secure embedded operating system which is built on top of the proven seL4 software ecosystem and consequently relies on trusted open-source components. TRENTOS provides high level libraries to provide developers functionality, such as logging for example, encapsulating the seL4 and CAmkES implementation underneath. This allows developers to fully focus on the creation of secure applications, and not worry about lower-level details of the underlying architecture. This especially alleviates the complexities of developing secure embedded systems.

TRENTOS consists of a set of building blocks, which are provided by a typical embedded OS for easing the development of custom applications. The framework's modular structure is structured as a set of libraries, written in the C programming language. The libraries provide all required core OS features (e.g. networking or storage facilities), logging capabilities, security primitives (e.g. cryptography) as well as additional helper functionality. The TRENTOS SDK further equips a developer with extensive documentation as well as all the tools required for building, testing and finally deploying a TRENTOS-based system to the real world.

TRENTOS also leverages the component character of CAmkES, which demands concise interface definitions to interact with the OS. TRENTOS therefore provides both a dedicated C API as well as a set of standardized interfaces utilizing the CAmkES IDL. A typical TRENTOS component is then able to use the TRENTOS API (e.g. the socket API) by either providing or consuming respective functionality via an RPC interface. A TRENTOS component is built on top of CAmkES facilities and therefore basically behaves like a CAmkES component; thus, it internally has to stick to the CAmkES architecture definition language (ADL) and therefore also requires additional programmer-provided component code. The TRENTOS SDK provides a set of standard components, which internally adhere to selected TRENTOS libraries. They provide a kind of reference implementation for typical OS functionality, being accessible via the TRENTOS API. Examples are provided in form of platform specific device drivers (e.g., a network driver for the Raspberry Pi 3 B+) as well as in form of intermediate layers (e.g. a network stack component).

4.2 THE FOUNDATION PART 2: A SOFTWARE FRAMEWORK FOR CERTIFIED SYSTEMS

RTI's interest in trustworthy systems was driven first by the avionics market, and then by the automotive market. We have several hundred customer projects alone tied to autonomous vehicles. Customers tend to start with our non-certified products for initial prototyping and development because they are more feature rich. Once they have matured their designs sufficiently, they make the jump to our software with commercial certification evidence available. You may also find value in taking a similar approach.

RTI Connext Cert® is a RTCA DO-178C DAL A connectivity framework that implements a flight safety subset of the DDS specification. It has also recently attained certification at the highest level for automotive systems – ISO 26262 ASIL-D. The cost of DO-178C certification alone

exceeded $200 per line of code, and the process took two years. We worked very closely with a certification house and used their tools to develop the high-level and low-level requirements (HLRs and LLRs) along with the other certification evidence that was needed.

As addressed earlier, a DDS solution with commercial certification evidence brings numerous benefits beyond just shortening development time and reducing overall system costs. It also offers scalability, cross-platform/language/vendor interoperability, performance, resilience, and modularity. RTI Connext Cert can also run on commercial RTOSs, including DDC-I Deos, Green Hills Integrity-178, Lynx Software LynxOS-178, Sysgo PikeOS, and Wind River VxWorks. RTI Connext Cert combined with a safety RTOS is an architectural option if your system requirements do not steer you toward a small footprint trusted microkernel.

We have been continuously involved with the seL4 microkernel since 2015 when we were awarded the first in a series of related research contracts with DARPA. Our overall goal has been to explore and develop a high assurance software stack for critical real-time systems. We have been building reference architectures and implementations to highlight design alternatives and related performance. This paper is part of our efforts to educate and to evangelize this information.

One example of our research focused on controlling a medical device (an infusion pump) – see Figure 4-1. The Infusion Pump Controller operated an Infusion Pump device via serial commands to an external step motor. The controller application ran on seL4, and it was deployed on a SabreLite iMX6 board. RTI Connext Cert listened for “infusion requests” coming from the remote authorized hosts, and periodically published status updates on the state of the pump to any application authorized to join the DDS databus. The “Root Task” is the process started by the seL4 kernel and responsible for the initialization of all required system services, before spawning the controller’s process, which access RTI Connext Cert services.

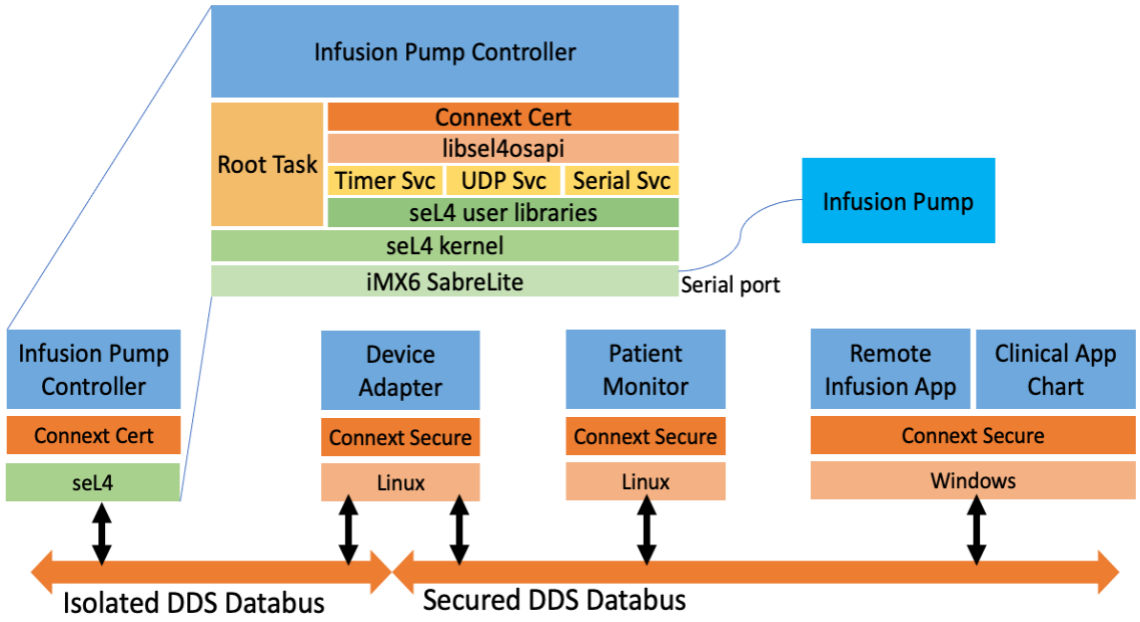

Figure 4-1. Infusion pump controller architecture.

When running directly on seL4, DDS requires a custom library to provide key services that seL4 does not support (libsel4osapi). When using CAmkES, which provides these services, this library is no longer necessary.

4.3 Example Trustworthy Architectures

Two generalized architectures that we have been utilizing include:

1. Running applications directly within the trusted user space of seL4 with no operating system
2. Treating seL4 as a hypervisor and running applications within virtual machines (VMs).

A third architectural option is to combine these. We discuss each below. An important advantage of using DDS is that your application code will not be dependent upon seL4 – it can run in either architecture, along with numerous other OS/HW combinations. These applications can communicate with all other DDS applications running anywhere on your computer, network, or WAN. Moreover, these applications could all be running on different OS/HW and written in different languages.

4.3.1 Running in seL4 Trusted User Space

The first option we present is to deploy your applications directly on seL4/CAmkES along with a certifiable DDS library (see Figure 4-2). As noted earlier, this option offers the most trustworthy solution. You can either utilize the CAmkES component framework or develop directly as a native seL4 library. In both cases, your applications would only need to invoke the interfaces provided

by DDS. The main benefit of using the CAmkES design pattern is that the developer gets verified interfaces "for free". This may lead to a much quicker verification process.

This architecture has three key advantages:

1. It avoids the need for a full-blown operating system,
2. it reduces the power, memory, and CPU requirements for your system
3. It significantly reduces the attack surface.

It also comes with three disadvantages

1. Your applications need to be written in C
2. The development environment is very limited
3. There is extra work you need to do in your application to bootstrap your code.

While you can copy from examples, all of this code will still need to be reviewed and adapted to your needs. In particular, there will be quite a bit of seL4 configuration needed, allocation of memory, bootstrapping the standard C library, bootstrapping all the I/O, etc. While CAmkES simplifies these bootstrapping steps, another approach will be needed if you want to avoid the first two disadvantages. We discuss this next.

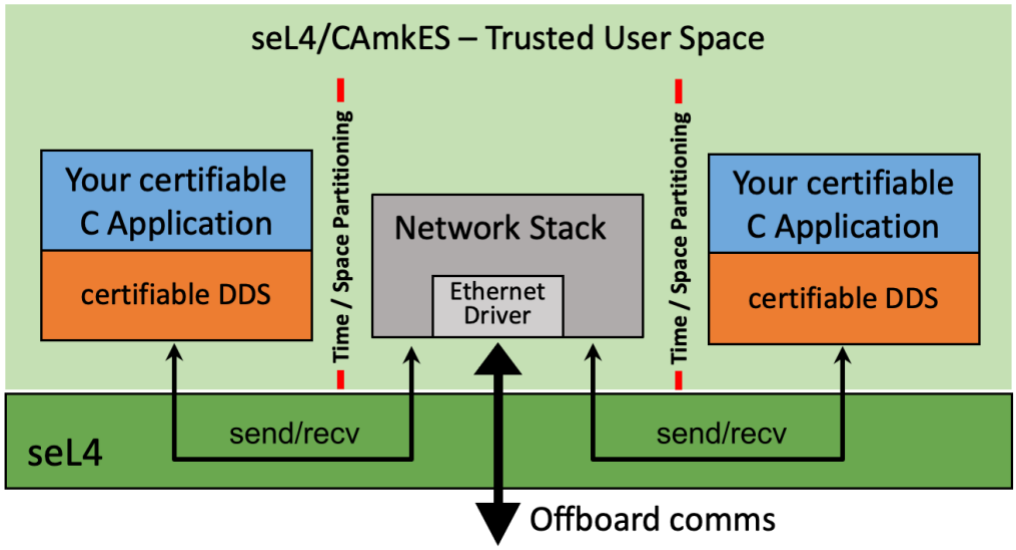

Figure 4-2. Example when running your applications directly on seL4/CAmkES in trusted user space.

4.3.2 seL4 as a Hypervisor

The second architectural option reaps the benefits of seL4 while avoiding the development constraints imposed by seL4's trusted user space. Using this approach, seL4 can be used as a hypervisor (see Figure 4-3). It provides separation guarantees to ensure that there is no unintentional information leakage between the VMs. While the Linux attack surface within each VM still exists, the isolation provided by seL4 will ensure that a failure in one VM will not bring down any other running VMs.

From a development perspective, the main benefit of this approach is that you have the freedom to build your applications in virtual machines, with all of the normal development tools and language options. You can also run other large or legacy applications that would be far too expensive to port to native seL4 and to formally verify. The use case for this design is to have little development done on seL4. This approach is being widely used in both defense and commercial industries.

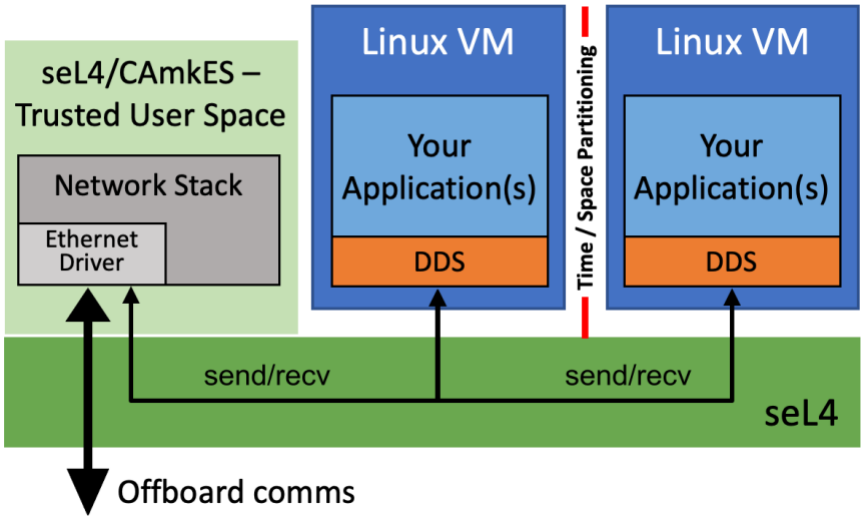

Figure 4-3. Using seL4 as a hypervisor.

This option is also used as an initial step to a migration of an existing system to seL4. The typical process to retrofit an existing system to run on top of seL4 is to start by separating trusted and untrusted components and isolating them into virtual machines and slowly migrating key components (trusted) into seL4 native applications. This may culminate in a third architecture option: a combination of some applications running in VMs and applications running directly in seL4 execution space.

5 CONCLUSION

The market demand for trustworthy systems is accelerating. However, this endeavor is not for the faint of heart. The expertise needed to build and certify these systems is limited. However, by building upon a high assurance stack, you can significantly reduce your time to market and your ongoing development and certification costs. Also, keep in mind that choices early on can have a significant impact on overall costs, particularly when it comes to certification. We recommend talking to a certification expert early on. We have included a list of several companies in the appendices.

We have been exploring ways to make this journey shorter, with lower costs, and more enjoyable. The high assurance software stack that we proposed is one option to consider. For DDS, seL4 creates an enriched, low cost, small footprint, high assurance alternative for our customers. For seL4, DDS provides an open standards-based communications protocol. DDS will

significantly simplify seL4 inter-component/application development, reduce associated costs, and promote component interoperability in the seL4 development community.

We have found that the certification cost and time can be significantly reduced with this approach, when using pre-certified sel4 and DDS components from vendors that offer certified stacks. Given that the cost of certification alone spans $100 to $300 (or more) per line of code, depending upon the certification level required, there is a strong motivation to find and use as many pre-certifiable components as you can in order to significantly reduce the program risk and cost and time to delivery.

6 GETTING STARTED

To get started, we have provided links to a number of general resources below about seL4, TRENTOS and DDS. We have also provided links to the commercial technologies that we used.

Please reach out if you are interested!

General Resources:

- Object Management Group Data Distribution Service (DDS). These links provide more information about what DDS is, a list of DDS vendors, and the standard itself.
    - DDS Foundation, *https://www.dds-foundation.org/*
    - OMG DDS Standard, https://www.omg.org/omg-dds-portal/
- seL4. These links provide in-depth information about seL4, and. access to the seL4 source code. There are two groups that actively support seL4:
    - seL4 Foundation, *https://sel4.systems/*
    - Trusted Computing Center of Excellence (TCCOE), *https://trustedcomputingcoe.org/*
- Software Certification standards
    - Flight Safety
        - RTCA DO178C
        - Introduction to the Certification process, http://www.verocel.com/wp-content/uploads/DO-178C-Presentation.pdf
    - Autonomous Vehicles
        - https://www.iso.org/standard/68383.html
    - Medical Devices (FDA)
        - *https://www.fda.gov/regulatory-information/search-fda-guidance-documents/content-premarket-submissions-management-cybersecurity-medical-devices*

- A starting point for Software Certification Companies. Others can be found via a Google search.
    - *AFuzion*, https://afuzion.com/
    - *AVISTA/Belcan*, https://www.avistainc.com/
    - *Kugler Maag*, https://www.kuglermaag.com/
    - *Mannarino*, https://www.mss.ca/
    - *Verocel*, https://www.verocel.com/

Vendor-specific:

- Hensoldt Cyber, a German company with focus on secure IT, develops TRENTOS https://hensoldt-cyber.com
- TRENTOS, a secure embedded OS framework based on seL4. https://www.trentos.de/
- RTI Connext Cert, https://www.rti.com/products/connext-dds-cert
- RTI's Research Team, *https://www.rti.com/company/research-programs*

7 REFERENCES

The Data Distribution Service Specification, v1.2, *http://www.omg.org/spec/DDS/1.2/*

The Real-Time Publish Subscribe DDS Wire Protocol, v2.1, *http://www.omg.org/spec/DDSI/2.1/*

The Data Distribution Service Security Specification,
https://www.omg.org/spec/DDS-SECURITY/

8 ACKNOWLEDGEMENTS

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) Small Business Innovation Research (SBIR) Program Office under Contract No. W31P4Q-20-C-0046. Approved for Public Release, Distribution Unlimited.

The views expressed in the IIC Journal of Innovation are the author's views and do not necessarily represent the views of their respective employers nor those of the Industry IoT Consortium®.

➢ Return to *IIC Journal of Innovation landing page* for more articles and past editions.