Harmonizing ARINC 653 and Realtime POSIX for Conformance to the FACE Technical Standard

Gedare Bloom
University of Colorado Colorado Springs
Colorado Spring, CO, USA
gbloom@uccs.edu

Joel Sherrill
OAR Corporation
Huntsville, AL, USA
joel.sherrill@oarcorp.com

***Abstract*—The avionics industry is converging toward the next generation of software standards produced by The Open Group via the Future Airborne Capability Environment (FACE) consortium and related FACE Technical Standard. The standard combines ARINC 653, a previous avionics standard, with subsets of POSIX 1003.1 that are closely aligned with the POSIX realtime profiles PSE52, PSE53, and PSE54. In this paper, we describe our approach to design, implement, and certify a system with FACE Conformance to the FACE Operating System Segment Safety Base profile. Our approach integrates the ARINC 653-compliant Deos with RTEMS, an open-source real-time operating system (RTOS). Our goal in combining Deos/RTEMS was to achieve certification of FACE Conformance in a low-cost manner by relying on existing, mature software that already provides the majority of the functionality required by the FACE Technical Standard. We reached our goal with under 10,000 source lines of code (SLOC) written to integrate RTEMS into Deos and implement any additional POSIX application programming interfaces (APIs) and tests needed for certification.**

***Index Terms*—Future Airborne Capability Environment, FACE, ARINC 653, POSIX, RTOS, RTEMS, Deos**

I. Introduction

The Future Airborne Capability Environment (FACE) Technical Standard—Edition 3.0 [1] as of this writing—is a standard for avionics platforms that is produced by The Open Group using Modular Open Systems Approach (MOSA) principles. The standard defines system software support and application programming interfaces (APIs) needed for portable avionics software in a component-based architecture consistent with an object-oriented real-time computing paradigm.

Four OS Segment (OSS) profiles are defined by the FACE Technical Standard: Security, Safety Base, Safety Extended, and General Purpose. These profiles are a superset of ARINC 653 Part 1 (revisions 3 or 4) [2] and subsets of POSIX that address the varying application-specific needs of avionics software. Security is the most restrictive environment having the fewest APIs available for applications, which may have little to do with cybersecurity as a general concept other than simplicity lending itself to security. An OSS certified for all the capabilities of a particular profile is said to have *conformance* for that profile. The General Purpose Profile is a POSIX-based execution environment that optionally may include ARINC 653, while the other three profiles require a mix of ARINC 653 and POSIX support from the OS.

This work is partially supported by NSF OAC-2001789. The authors also thank DDC-I, Inc. for their role in this effort.

In this paper, we describe our approach and experience developing an OS aiming at FACE conformance for the Security, Safety Base, and Safety Extended profiles. One challenging aspect for FACE conformance is that prior standards have not required simultaneous POSIX and ARINC 653 support. Thus, prior real-time operating systems (RTOSs) have focused on one or the other. We identified that the requirements of the FACE Technical Standard logically define two execution environments—one supporting POSIX and one for ARINC 653—and that nothing in the standard restricts using two OSs to satisfy the requirements. As a result, we chose a path of least resistance that combined Deos, an RTOS with ARINC 653 support, with RTEMS, an open-source RTOS with POSIX support. At a high-level our approach mimics paravirtualization with Deos as the host and RTEMS as a guest.

Despite the separation of ARINC 653 and POSIX functionality between Deos and RTEMS, a POSIX thread can use a subset of the ARINC 653 APIs. In addition, multiple partitions within Deos can contain RTEMS instances, and therefore can execute multiple POSIX (RTEMS) applications concurrently, which is logically equivalent to execution of multiple statically-created POSIX processes.

II. Background

A. Future Airborne Capability Environment (FACE)

The FACE Consortium was founded by The Open Group in 2010 to define an open architecture for avionics. Since its founding, three major editions of the FACE Technical Standard and a wide variety of technical and business supporting documents have been released. Edition 2.1 of the standard was released in May 2014, and Edition 3.0 in November 2017. An Edition 3.1 corrigenda is expected in 2020 that incorporates corrections for identified issues.

As depicted in Fig. 1, the FACE Reference Architecture was introduced in Edition 1.0 with five core segments that remain to this day: Operating System Segment, Portable Components Segment, Platform-Specific Services Segment, I/O Services Segment, and Transport Services Segment. The basic functionality of each segment is the same over time although the interfaces have undergone significant changes. Editions

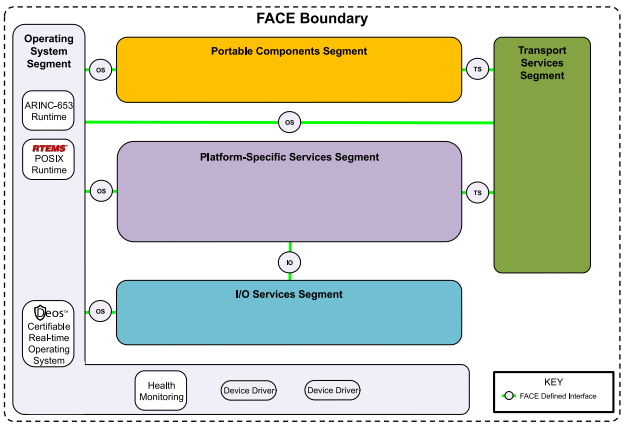

Fig. 1. FACE Architecture Diagram

1.x and 2.x use procedural-style interfaces, but Edition 3.x uses an object-oriented approach for interfaces defined by the FACE Consortium. Edition 2.0 adopted Interface Definition Language for describing the interfaces in addition to introducing a new interface: FACE Health Monitoring and Fault Management. Edition 3.0 introduced the Configuration, Life Cycle, and Component State Persistent interfaces.

The Operating System Segment incorporates existing OS, programming language, and graphics standards based on their prior or anticipated use in safety-critical avionics system. Relatively few changes have been made to the Operating System Segment across editions of the FACE Technical Standard as a result of relying on established standards. The POSIX profiles for Editions 1.x and 2.x are identical. Edition 3.0 added the POSIX `clock_nanosleep` method and allowed for optional replacement of the ARINC 653P1-3 by ARINC 653P1-4 (revision 3 by 4) [2]; the latter was released in August 2015. The upcoming Edition 3.1 includes changes to the POSIX profiles that allow POSIX multi-process APIs to be optional in the Safety Extended and General Purpose Profiles and increase alignment with the profiles defined by the Software Communications Architecture [3] standard, which is used primarily for software-defined radios.

B. Deos

Deos is a safety-critical real-time operating system with time- and space-partitioning capabilities that is certifiable to the most stringent DO-178 Design Assurance Level (DAL) A. Out-of-the-box Deos already contains all supported APIs needed from ARINC 653 for achieving FACE conformance. Deos supports the x86, PowerPC, MIPS, and ARM architectures.

C. RTEMS

RTEMS is an open-source hard real-time operating system that was first produced in the 1980s in conjunction with the US Army. RTEMS has one of the most POSIX-rich environments among open-source RTOSs, and it is therefore an appealing choice for use in applications aligned with the FACE Technical Standard or other POSIX-based standards. RTEMS supports many architectures including those supported by Deos, and has robust support for multicore scheduling and synchronization. RTEMS is a single address space operating system (SASOS) with a flat memory model that historically has been statically linked with application code in a single process, multi-threaded environment with no spatial isolation.

III. Methodology

In this section we describe the approach we took to align Deos/RTEMS with the FACE Technical Standard. The main motivation for our approach is that ARINC 653 and POSIX services need not be provided by the same OS environment. Thus, we use paravirtualization techniques to support executing Deos and RTEMS in tandem to provide both sets of services. Applications that use ARINC 653 or POSIX can run under Deos or RTEMS, respectively.

A challenge for our approach is that Deos is not designed to act as a hypervisor and so it does not have an existing set of hypercalls. The design of Deos as a time- and space-partitioning RTOS however does permit running RTEMS inside a partition, which yields a similar result as if RTEMS were in a virtual machine environment provided by a hypervisor. We therefore implemented a thin *adapter layer* that uses traditional Deos syscalls like a hypercall interface and provides RTEMS with access to request supervisor services.

A. RTEMS Paravirtualization

Paravirtualization of RTEMS has been an ongoing effort spanning nearly a decade and several development projects related to executing RTEMS efficiently as a guest within a host (hypervisor or partitioning OS) environment [4]. This effort produced an initial framework for defining a paravirtualized RTEMS guest. We refined this framework and generalized it to be host-agnostic. The general framework has been adopted within RTEMS and is used by others to run RTEMS in a virtualized environment.

Our approach for paravirtualization logically follows RTEMS' layered software architecture. We use a compile-time option (RTEMS_PARAVIRT) to control a kernel build for paravirtualization. This option avoids the use of privileged and sensitive instructions/registers in the architecture-specific portion of RTEMS. Often, such usage is replaced with function calls that need to be implemented to make the appropriate call to the host OS (i.e., hypercall/syscall). These functions are defined within an RTEMS board support package or adapter that provides support to run RTEMS within a virtual environment as opposed to a physical board.

We provided initial support to paravirtualize RTEMS for the x86, ARM, and PowerPC architectures. Others have paravirtualized the SPARC v7/8 architecture. Our work targeted Deos, while subsequent work has resulted in support for the XtratuM and Xen hypervisors.

B. Time Management

We faced two challenges related to time. First is that when RTEMS is scheduled as a guest it does not understand the

passage of time outside its own execution, and therefore the system clock is not accurate. Second is the potential need to synchronize the time-of-day (date/time) in an RTEMS partition with time changes made in a different partition. We discuss both of these challenges and our solutions in the following.

1) System Clock: The primary difficulty for tracking time is that the time that elapses while the RTEMS partition is not executing does not get relayed to the partition. Furthermore, the partition does not know when it is context switched. Thus, the start and end of a partition's scheduling window is transparent to the executing thread. We adopt a simple approach to identify the start of the window by causing Deos to deliver the RTEMS partition a time budget exceeded exception at the start of each window. By never yielding (self-suspending) back to Deos from RTEMS we ensure that this exception will always be delivered like a software interrupt aligned with the start of each new scheduling window, i.e., each activation of the RTEMS partition. We register a modified clock tick interrupt handler to handle the exception; thus, each scheduling window denotes a set of clock ticks.

The clock tick handler—driven by the time budget exception—implements time management in a relatively straightforward manner. We rely on Deos' uptime (time offset since boot) as a free-running clock source to keep track of the passage of time. The exception handler fetches the uptime with a syscall, calculates the time that elapsed since the start of the previous scheduling window. From this time interval and the configured system clock frequency, the handler calculates how many clock ticks have elapsed. Any remainder of time for a partial tick from the preceding window gets added to the current interval, and any partial tick left at the end will get passed to the subsequent window. The handler then invokes the `rtems_clock_tick` method the same number of times as full clock ticks that were calculated. This method is the interface in RTEMS that clock drivers use to advance the tick-based system clock. The number of calls made can vary depending on the length of time between the start of partition windows.

2) Time-of-Day Synchronization: Another time-related challenge is communicating time changes between guest and host environments. The FACE Technical Standard requires that the ability to set the date/time is configurable for partitions, and that if a partition sets the time then the new time gets set for all partitions. Deos already supports the configuration of a partition's permission to set the global time, thus satisfying the former requirement. We satisfied the latter requirement in both directions by augmenting RTEMS to allow for time updates from an external source, and by adding a server thread in RTEMS that can synchronize the RTEMS partition's time with Deos' time.

The modification to synchronize time with an external source is a general improvement to RTEMS that we have merged with the upstream public repository. Our approach essentially introduces an event registration mechanism for registering a callback function in case a time change (set) happens inside RTEMS. We use this mechanism to propagate time changes from an RTEMS partition back to Deos; however, the callback could also be useful for example to update a hardware (real-time) clock when the software time changes. One might use such an approach in combination with a network time synchronization protocol.

The server thread that synchronizes RTEMS time to Deos is an optional feature that must be enabled by the application configuration. When enabled, the system startup invokes `deos_tod_synchronization_initialize`, which creates the server thread and obtains the current Deos uptime and time *bias*. This bias is the difference between uptime and the current time-of-day. The clock tick handler—running at the start of each execution window—checks if the bias value has changed since its previous reading and, if so, unblocks the server thread to synchronize with the updated time-of-day. The server thread calculates the current time from the Deos uptime plus bias and calls `clock_settime` to update the local partition's notion of the time-of-day.

C. Thread Stack Allocation

RTEMS normally allocates thread stack memory from a memory pool which is located in memory between the end of the loaded binary image and the end of physical memory. Under typical use, this allocation is non-problematic because RTEMS operates without a memory management or protection unit (MMU/MPU); thus, the memory "hole" between the end of the binary image and physical memory is always available for the OS to utilize. Deos provides spatial isolation with MMU support to provide each partition with protected regions for code, read-only data, writeable data, and thread stacks. Deos initializes the stack memory region in the MMU to ensure stack pointer accesses remain within the region allocated for thread stacks. Thus, RTEMS needs to be aware of the allocated region for stack allocation. RTEMS already allows an application to configure its own thread stack allocator and deallocator, so we provided a custom stack allocator in the adapter layer that reuses the RTEMS code for the heap memory.

D. Memory Management

One aspect of POSIX support that is absent from RTEMS is in the area of virtual memory and memory management services. These services have not been needed by RTEMS applications in the past. However, all the FACE OSS profiles require support for `shm_open` and `mmap` methods, and although only required in the General Purpose Profile we also found it natural to provide `shm_unlink` and `munmap` support. (Our system does not support memory (page) locking or protection, which are only required in the FACE General Purpose Profile.)

We implemented support in RTEMS for both shared memory objects and memory mapped files in a way that allows applications to use those APIs within RTEMS. The RTEMS implementation by itself is not overly interesting, because RTEMS still does not support virtual memory so calls to `mmap` are satisfied primarily through memory aliasing or memory

copying. We extended this implementation to call-out to the Deos adapter to use Deos' memory management routines to provide inter-partition support for shared memory objects and memory mapped files, which does provide for more feature-rich memory services beyond the basic service in RTEMS alone.

Subsequently to our effort implementing `mmap` there has been interest in the RTEMS community for using and improving the support it provides. The main driver for this interest seems to be the ability to use code from other monolithic kernels (i.e., FreeBSD) for device drivers that use `mmap` to access device memory.

E. Adding POSIX APIs

With respect to the POSIX API requirements, the FACE profiles align with the POSIX Realtime System Profiles defined in POSIX.1-2003 (IEEE Std 1000.13-2003 [5]): Minimal Realtime System Profile (PSE51), Realtime Controller System Profile (PSE52), Dedicated Realtime System Profile (PSE53), and Multi-Purpose Realtime System Profile (PSE54). Table I summarizes the alignment of the POSIX Realtime profiles with closely matching FACE profiles; we define the aligned metric for how closely they match as the number of methods shared in both profiles divided by the number of methods in the FACE profile. That is, we ignore the excess of methods that are in the POSIX profile but were omitted in the FACE profile. Many of the omitted methods were dropped in FACE profiles due to those methods being deprecated or considered unsafe or insecure to use in common programming recommendations. Note that the FACE OSS profiles were defined following POSIX.1-2008, while the PSE profiles have not been updated since they were included in POSIX.1-2003.

The PSE51 profile is not a good match for any FACE profile. The FACE Safety Base is a subset of PSE53 except for the addition of `posix_devctl`, `pthread_mutex_timedlock`, and `umask`; it is also a close match to PSE52, but with networking APIs. Thus, an OS with PSE53 support could simply add the three additional API calls to support the Safety Base Profile. FACE Safety Extended is a subset of PSE54 with the addition of `alarm`, `kill`, `pause`, `posix_devctl`, `pthread_mutex_timedlock`, and `waitpid`. The General Purpose Profile is a subset of POSIX.1-2008 (IEEE Std 1003.1 2008 Edition) with the addition of `posix_devctl`, thus an OS that is compliant with POSIX.1-2008 would only need to add `posix_devctl` to support every FACE OSS profile. Note however that the FACE profiles are subsets of POSIX, so, for example, it is possible to have support for Safety Extended without having full support for PSE54; indeed, there are 576 APIs in PSE54 that are not in Safety Extended. An OS may optionally provide additional APIs beyond the standard, but an application is restricted to calling only the APIs available in the profile it uses.

The outlier in alignment between POSIX and FACE standards is the `posix_devctl` method, which is a standardized alternative to `ioctl` that comes from POSIX 1003.26 [6]. The FACE Technical Standard requires only limited support for `posix_devctl` for use on non-blocking sockets. In a brief review we looked at several POSIX-compliant OSs, including Linux and FreeBSD, and found that none of them included support for the `posix_devctl` method; we did find some systems that supported the `devctl` method, which is essentially the same as `posix_devctl` but was in a draft standard that was never standardized and eventually deprecated. We also did not identify any required use cases for this method outside of the FACE Technical Standard. We added support for `posix_devctl` method in RTEMS as a simple wrapper around `ioctl`, which could be done by any OS that has a (sane) version of `ioctl` already in place. To date however we are not aware of any other open-source implementations of this standard API method.

F. Networking with lwIP

Lightweight IP (lwIP) is an open-source networking stack that is commonly used in embedded systems because of its low spatial resource overhead. The lwIP stack is also appealing for safety qualification because it is a relatively small codebase that can be configured to remove features and thus reduce the complexity for qualification. Deos uses lwIP, but we found it needed to be updated and have more features enabled to satisfy the networking requirements of the FACE Technical Standard. By default RTEMS does not use lwIP, although lwIP has been ported to run with RTEMS as an alternative network stack. Instead, RTEMS has a networking stack that is derived from FreeBSD sources and headers. For implementing the networking APIs, we use the existing FreeBSD-based networking header files and implemented a client in RTEMS that maps the networking APIs to lwIP's interfaces. This client uses a remote procedure call to invoke lwIP on a server thread that executes in another partition. The client supports concurrent requests from multiple threads, and the server supports multiple concurrent client requests.

G. Joint ARINC 653 and POSIX

The FACE Technical Standard requires that, if ARINC 653 is supported, the ARINC 653 services for sampling ports, queuing ports, and health services must be available in a POSIX environment. Rather than have two distinct implementations of these services, the Deos implementation of ARINC 653 services was refactored and modified to allow adaptation to a particular run-time. A set of plugins are instantiated when a partition is created depending on whether it is a POSIX or ARINC 653 partition. These plugins allow the partition to bind to a common implementation—within a Deos shared library that is statically loaded at boot-time—for run-time services such as thread state management (blocking/unblocking), synchronization, and scheduling. When a task blocks in an ARINC 653 service, the task will be moved to a queue of either ARINC 653 or POSIX threads, depending on how its partition was initialized. As a result, software runtime support for POSIX and ARINC 653 applications remains modular and, importantly from a certification standpoint, the underlying

TABLE I
ALIGNMENT OF FACE TECHNICAL STANDARD 3.0 AND POSIX REALTIME PROFILES. ALIGNMENT PERCENTAGE IS THE RATIO OF SHARED METHODS IN BOTH PROFILES DIVIDED BY THE NUMBER OF METHODS IN THE FACE PROFILE.

POSIX Profile	Closest FACE Profile	# Shared Methods	Omitted in FACE Profile	Added in FACE Profile	Alignment
PSE51	Security	111	174	53	67.7%
PSE52	Security	140	489	24	85.4%
PSE52	Safety Base	216	413	31	87.5%
PSE53	Security	162	157	2	98.8%
PSE53	Safety Base	244	508	3	98.8%
PSE53	Safety Extended	319	433	17	94.9%
PSE54	Security	160	746	4	97.6%
PSE54	Safety Base	243	665	4	98.4%
PSE54	Safety Extended	330	578	6	98.2%
PSE54	General Purpose	789	119	23	97.2%
Std 1003.1	General Purpose	811	314	1	99.9%

shared library is identical for both APIs which allows reuse of evidence and history of the binary library artifacts.

H. Development Tools

The development tools to work with Deos are integrated in the OpenArbor IDE, which is an Eclipse-based development environment produced by DDC-I, the company that also makes Deos. The OpenArbor IDE included support for obtaining a timeline of the execution of ARINC 653 processes. The timeline is constructed from a log that captures the time when each context switch occurs and the previous/next executing threads. We leveraged an existing capability in RTEMS that allows adding callouts to hooks invoked at specific points in the thread lifecycle. In particular, we provide an extension set in the adapter for Deos that provides a callout for the context switch event to log the timestamp and two threads—executing and heir in RTEMS parlance. The presence of this hook made thread timeline visualization surprisingly simpler than we initially anticipated.

Another developer aid available in the OpenArbor IDE is a mechanism to display the set of ARINC 653 processes in a partition along with information about each process's attributes such as its id, priority, allocated stack location and size, and stack space usage. This information is provided by an introspection monitor executing in a super-privileged partition that has permission to examine memory in every partition. RTEMS thread-specific support was developed for the introspection monitor. The main challenge to implement this support was that the interface to the partition was restricted to 32-bit load word operations. Access to fields in structures required using `offsetof` and addition, and pointers had to be cast and dereferenced explicitly. The id, priority, and stack space allocated were obtained directly by finding and parsing the internal RTEMS data structure representing a thread control block. Information about the stack space usage however is not tracked by default in RTEMS. To get this information, we enabled a stack checker capability, which is optional in RTEMS, that populates the stack region with a fixed value (i.e., a repeated canary). We then get an estimate of the stack usage by walking the stack and finding the first canary value from the start.

I. Cross-OS Configuration

Deos was designed to support statically configured safety-critical systems amenable to safety certification and qualification processes. As such, Deos has to be configured *a priori* with the number of partitions, their characteristics, and any I/O or IPC connections they have. The execution of these partitions is manually scheduled by assigning individual partitions to execution time windows. Partitions may yield their time window early with their remaining time donated to a pool of slack time; a set of partitions may be associated with slack time. In multiprocessor configurations, partitions can be assigned to specific cores during a time window or left idle. This manual assignment of partitions to time windows allows for detailed offline analysis of the execution including worst-case execution time (WCET) analysis.

Many of Deos's configuration parameters apply to a partition independent of whether the partition contains an ARINC 653 or RTEMS application. However, we added awareness of some of these parameters in the adapter layer of RTEMS to support better integration with Deos, including:

- ARINC 653 queuing ports;
- ARINC 653 shared memory regions;
- Permission to set the global time;
- Free memory space available;
- Video buffer row and height position.

The first two of these parameters are provided to RTEMS at compile-time as they are statically configured parameters in Deos. The configuration setting for permission to set the global time is obtained through a series of syscalls to Deos that probe for descending access permission. Once the access permission is determined it is used to control whether and how RTEMS should synchronize its notion of time with Deos as described in Section III-B2. The remaining configuration parameters are parsed from a boot-time string that is passed from Deos to the partition as a list of key-value pairs. The amount of free space available is used to set up the dynamic memory allocators for the C Program Heap and the RTEMS Workspace; the latter is the area from which RTEMS allocates internal OS objects. The video buffer is a virtual 80x25 console that can be shared among multiple partitions. Architectural-specific boot-

time parameters are passed through the same string, e.g., the x86 adapter has boot parameters to allow using COM1 as a console port and to choose between a soft-reset or halt when the application exits.

IV. Evaluation

We evaluate the extent to which we have been successful along two primary axes: conformance to the FACE Technical Standard, and programmer effort measured with source lines of code (SLOC).

A. FACE Conformance

The FACE Consortium maintains a conformance process that ensures that Units of Conformance (UoCs) do meet the FACE requirements as claimed. Once a UoC is deemed conformant, it may be listed in the FACE Registry [7]. Conformance is a Boolean condition: there is no concept of partial conformance.

The FACE Conformance process requires examination of more than code. A FACE Verification Authority (VA) is presented with a Software Verification Package that provides evidence such that the VA can independently verify that FACE requirements for the software have been met. The Software Verification Package includes documentation that addresses all the potentially applicable FACE requirements. Some requirements are not applicable, e.g., an Operating System Segment UoC may optionally provide a Java runtime, but the Deos/RTEMS environment does not, and so there must be documentation for the VA to know that this optional requirement is not being addressed. For functional requirements, there must be documented test cases that demonstrate the requirement is being met, which the VA will check against a Software Test Report for passing test results.

A (proprietary) mapping document was developed that provides the location of specific evidence in Deos and RTEMS artifacts along with the relevant test cases in the Software Test Report for the VA to check that Deos/RTEMS satisfies the FACE Technical Standard requirements. This document is specific to a version of the standard and the product containing the evidence. For many cases of the requirements, we could rely on the existing standards that FACE includes to point where in Deos or RTEMS the evidence already existed.

The Deos/RTEMS product was certified FACE Conformant for the Safety Base OS Profile to the FACE Technical Standard 3.0 in September 2019. This certification also covers the Security Profile, because it is a subset of Safety Base. Conformance was achieved for the PowerPC architecture; other architectures are planned to go through a streamlined process for certifying conformance of variants that rely on already-certified artifacts. At the time it achieved conformance, Deos/RTEMS was the 10th FACE Operating System Segment to be certified; DornerWorks Virtuosity [8] was the 9th, and the first 8 were various products derived from the Integrity and VxWorks 653 commercial RTOSs.

TABLE II
SLOC Written for RTEMS to Support Deos/RTEMS

Category	SLOC	Work-Months (est.)	Cost (est. USD)
Kernel (Upstreamed)	1048	2.52	$28,381
Arch. Ind. Adapter	3037	7.59	$85,459
i386 (x86) Adapter	141	0.31	$3,454
PowerPC Adapter	99	0.21	$2,383
ARM Adapter	5	0.01	$104
Tests (Upstreamed)	647	1.52	$17,104
Adapter Tests	3813	9.78	$110,147
Total Upstreamed	1695	4.04	$45,485
Total Adapter	7095	17.9	$201,547
Total	**8790**	**21.94**	**$247,032**

B. Effort: Source Lines of Code (SLOC)

We used `sloccount` (v. 2.26) [9] to estimate the programmer effort and cost for the software written for Deos/RTEMS. This estimate focuses only on the software for RTEMS, because the modifications in Deos were not required to achieve FACE Conformance but rather were made as improvements to update software that was complementary to the FACE Technical Standard, e.g., the lwIP stack.

Table II shows a breakdown of the SLOC for the RTEMS code that was written to support FACE Conformance of Deos/RTEMS. We divide the code in three sets of categories: kernel code, adapter code, and tests. Furthermore, the adapter is partitioned into the architecture-independent code and the code that has been written to support specific instruction set architectures (ISAs). To date, we have completed support for the ARM, i386 (x86), and PowerPC ISAs, which account for the bulk of the avionics market. The kernel code and their tests have all been upstreamed, i.e., merged with the RTEMS Project's public Git repository. The adapter code and tests remain proprietary and need to be licensed (with Deos) to be useful.

The adapter code is largely independent of the target architecture. The PowerPC adapter consists mostly of some interrupt handling code, while the most complex adapter is the i386 one, which has extra support needed to use a COM port as a serial console. In the ARM case, we only have a few SLOC to set up the compiler and linker to use the architecture-independent sources; the effort to add ARM support was minimal. The bulk of the code is in the architecture-independent adapter and its tests that together constitute 6,850 SLOC, which is about 78% of the total SLOC.

V. Discussion

The primary motivation for the Deos/RTEMS effort was driven by the commercial factors involved in satisfying the joint ARINC 653 and POSIX requirements imposed by the FACE Technical Standard. Historically, ARINC 653 systems did not need POSIX, and therefore Deos had minimal support for POSIX APIs. RTEMS has not been oriented toward avionics applications and therefore neither had ARINC 653 support nor offered competition to Deos' market. Thus, the decision

to combine Deos with RTEMS was made as a synergistic opportunity for both products.

As both RTEMS and Deos are mature OSs, one of our key goals was to minimize changes to either. RTEMS had initial support for paravirtualization which allowed it to avoid privileged instructions such as interrupt disable/enable. This support was necessary to host RTEMS in an unprivileged address space container, but it was not sufficient to make RTEMS a fully integrated guest. Much of our effort focused on better integration via hypercalls/syscalls in the adapter layer to enable use of inter-process communication (IPC) mechanisms such as sampling and queuing ports, Deos lwIP network stack, and shared memory. This integration required matching the ARINC 653 view of IPC with that of the more flexible POSIX-based RTEMS environment.

Although POSIX support could have been implemented within Deos, it was viewed as beneficial to rely on the mature POSIX implementation available in RTEMS, which has flight heritage in space systems, and focus effort on certification rather than on a clean-slate POSIX implementation plus certification. In the end, this decision is validated by the achievement of FACE Conformance with relatively minor improvements made in Deos and reasonable effort (under 10000 SLOC) in RTEMS.

In an orthogonal yet broader view, the changes that were made to RTEMS and published in the open should facilitate other, similar efforts to use RTEMS in partitioned kernels and hypervisors as a guest for POSIX-based applications. Throughout the process of creating Deos/RTEMS and working toward FACE conformance we have made every effort to return code to the public open-source RTEMS Project that is generally useful and does not require a license for Deos. The changes made in RTEMS have been maintained as patches that apply on top of the public code base. We have attempted to merge back changes made in common code, i.e., outside of the adapter needed to interface with Deos. As shown in Table II, this code amounts to 1695 SLOC or approximately 19% of the cost/effort of the programming portion of the effort to achieve FACE conformance with Deos/RTEMS. Much of this code advances the available support of open-source software to achieve POSIX compliance. Thus, our work with Deos/RTEMS can play a wider role in impacting industry.

As a side-effect of seeking FACE Conformance, we developed a new, public RTEMS document, the RTEMS POSIX 1003.1 Compliance Guide [10]. This guide documents which standards-based APIs are supported by RTEMS in aggregate and broken down to specific standards such as POSIX.2003, POSIX.2008, C99, and C11. We also included the POSIX profiles defined by standards such as the FACE Technical Standard and the Software Communications Architecture [3]. We originally generated the bulk of this guide from an API tracking spreadsheet with a shell script that we published to the open-source RTEMS community. Subsequently, a member of that community rewrote our script to use Python and improved it. The data, program, and resultant guide are open-source and publicly adopted by the RTEMS Project.

Time management was a particularly challenging area because ARINC 653 environments have a static execution order of partitions with no need for time to advance while a partition is executing. ARINC 653 also has no notion of time-of-day (TOD). We discussed and prototyped multiple approaches before settling on the current approach for managing relative time inside each RTEMS partition, which is described in Section III-B2. Our penultimate approach put all the time management in the time budget exceeded exception handler, but we were concerned with the implications of TOD synchronization—particularly, calling `clock_set` from within an exception handler. After we split the time management into an exception handling component and a server (essentially equivalent to top- and bottom-half interrupt handling), our final approach had the added benefits of supporting nanosecond granularity timestamps and accurate CPU utilization statistics within RTEMS. Support for TOD itself required additions to Deos that required multiple design iterations to avoid any need for asynchronous notifications of time changes. Our decision to detect time changes and process clock ticks in RTEMS only at the beginning of a partition's execution window is similar to how input is expected to either be present or not at the beginning of a window.

Ultimately, the approach we have taken for the RTEMS partitions is reminiscent of logical execution time (LET) [11]. The start of each window accumulates the progression of time and reads input from IPC channels. The output of the partition is delayed (by idle thread execution) until the end of the execution window. Although we were not initially inspired by the LET paradigm, we found the eventual alignment of interest.

VI. Related Work

Bloom et al. [4] describe an effort to align the Deos and RTEMS OSs to the FACE Technical Standard 2.1 Safety Base Profile using a paravirtualized RTEMS executing as a partition to satisfy the POSIX services needed by FACE. The authors describe an approach for time and clock management, and the implementation of several methods from POSIX that were included in the Safety Base Profile but missing from RTEMS. However, this prior work did not implement all of the required POSIX APIs needed for the FACE Safety Base Profile, did not complete the integration of RTEMS with Deos, and did not describe any metrics for success or evaluation. Our work goes beyond this prior work with a different approach for time management, implementation of all POSIX methods needed by FACE Safety Base, a functionally complete implementation and integration with Deos including seamless IPC, kernel configuration, and developer tool support, and evaluation of our effort in terms of SLOC and certification of FACE Conformance (for Edition 3.0).

VanderLeest [8] describes preliminary work with Virtuosity as an effort to extend the Xen hypervisor with support for ARINC 653 and create a system aligned with the FACE Technical Standard 2.1. The author uses Linux as the system domain (dom0) and also as the guest to support POSIX-based

applications. Much of their motivation and approach is similar to ours, however the published work is not complete and they also mention concerns with using Linux due to its large code base. They do describe a rough cost estimate for the work they completed, which included "a few hundred" SLOC to add ARINC 653 scheduling in Xen and roughly 29,000 SLOC for IPC. Their goal is to provide a complete open-source software stack, and there are no commonly used and maintained open-source ARINC 653 software projects, so the implementation cost is expectedly higher than our approach relying on an existing, commercial ARINC 653 RTOS.

Zuepke et al. [12] describe AUTOBEST as a system that can support both ARINC 653 and the AUTOSAR standard for automotive software. Thus, AUTOBEST represents another point in the design space that offers support for multiple standards concurrently. The rationale given by the authors for combining these two standards is that the strong temporal partitioning of ARINC 653 is expected to yield benefits in automotive systems that centralize and combine multiple applications that previously executed on dedicated electronic control units. The approach taken by the authors involves customizing a common microkernel core for thread management and accommodating for differences in the standards most notably by using userspace libraries to implement synchronization. They discuss their decision to use a microkernel instead of a hypervisor with the rationale primarily based on the need in automotive standards (OSEK, AUTOSAR) to deliver prioritized interrupts. Our approach is different in several ways including that we take a virtualization-based approach and we are not considering automotive standards.

Crespo et al. [13] describe the XtratuM hypervisor, which implements a separation kernel for integrated modular avionics with an interface that is similar to ARINC 653. This interface facilitates integration of ARINC 653-compliant guests. XtratuM has also been used to run RTEMS as a guest. However, to the best of our knowledge XtratuM has not been used toward achieving FACE Conformance.

Rufino et al. [14] describe the AIR Partition Management microkernel that supports ARINC 653 by translating API method calls into an upcall to a POSIX or RTOS native API. Our approach differs in that we aim at FACE Conformance and we rely on IPC mechanisms instead of upcalls.

VII. Conclusion

In this paper, we have described the design and implementation of Deos/RTEMS, which is an OS environment that has been certified FACE Conformant aligned with the FACE Technical Standard 3.0 for the PowerPC architecture. The primary contribution of this work is to show that a relatively low-cost approach can satisfy a standard that includes multiple standards as subsets. We have also made contributions to the open-source ecosystem of the RTEMS Project that could filter to other industrial settings and platforms aside from the exemplar Deos/RTEMS.

Future work may re-certify with some software updates, certify other architectures (e.g., ARM and x86), and certify for past and forthcoming editions of the FACE Technical Standard. Additionally, the approach to use a hypervisor in combination with paravirtualized guests could benefit from further enhancements in real-time analysis and related techniques to allay concerns that the additional layer makes WCET and schedulability analysis too difficult. Despite academic efforts [15]–[17], commercial and open-source hypervisors often neglect real-time latency concerns in the hypervisor, and real-time embedded systems have a strong need for device and I/O virtualization that add further complications.

References

[1] "Technical Standard for FACE (Future Airborne Capability Environment), Edition 3.0," Nov. 2017. [Online]. Available: https://publications.opengroup.org/c17c

[2] "ARINC Specification 653 P1-4, Avionics Application Software Standard Interface, Part 1: Required Services," 2015.

[3] "Software Communications Architecture Specification," Joint Tactical Networking Center (JTNC), San Diego, CA, Tech. Rep. Version: 4.1, Aug. 2015.

[4] G. Bloom, J. Sherrill, and G. Gilliland, "Aligning Deos and RTEMS with the FACE Safety Base Operating System Profile," *SIGBED Rev.*, vol. 15, no. 1, pp. 15–21, Mar. 2018. [Online]. Available: http://doi.acm.org/10.1145/3199610.3199612

[5] "IEEE Standard for Information Technology- Standardized Application Environment Profile (AEP)-POSIX Realtime and Embedded Application Support," *IEEE Std 1003.13-2003 (Revision of IEEE Std 1003.13-1998)*, pp. i–164, 2004.

[6] "IEEE Standard for Information Technology - Portable Operating System Interface (POSIX(R)) - Part 26: Device Control Application Program Interface (API) [C Language]," *IEEE Std 1003.26-2003*, pp. 1–46, Sep. 2004.

[7] "FACE UOC Registry," Jan. 2020. [Online]. Available: https://www.facesoftware.org/registry

[8] S. H. VanderLeest, "Designing a future airborne capability environment (FACE) hypervisor for safety and security," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, Sep. 2017, pp. 1–9, iSSN: 2155-7209.

[9] D. A. Wheeler, "SLOCCount," Jan. 2020. [Online]. Available: https://dwheeler.com/sloccount/

[10] "RTEMS POSIX 1003.1 Compliance Guide (5.b12e82d). RTEMS POSIX 1003.1 Compliance Guide 5.b12e82d (19th December 2019) documentation," Dec. 2019. [Online]. Available: https://docs.rtems.org/branches/master/posix-compliance/index.html

[11] C. M. Kirsch and A. Sokolova, "The Logical Execution Time Paradigm," in *Advances in Real-Time Systems*, S. Chakraborty and J. Eberspcher, Eds. Berlin, Heidelberg: Springer, 2012, pp. 103–120.

[12] A. Zuepke, M. Bommert, and D. Lohmann, "AUTOBEST: a united AUTOSAR-OS and ARINC 653 kernel," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, Apr. 2015, pp. 133–144, iSSN: 1545-3421.

[13] A. Crespo, I. Ripoll, M. Masmano, P. Arberet, and J. Metge, "Xtratum an open source hypervisor for tsp embedded systems in aerospace," *Data Systems In Aerospace DASIA, Istanbul, Turkey*, May 2009.

[14] J. Rufino, J. Craveiro, T. Schoofs, C. Tatibana, and J. Windsor, "AIR Technology: a step towards ARINC 653 in space," in *Data Systems In Aerospace DASIA*, Istanbul, Turkey, 2009.

[15] Z. Jiang, N. C. Audsley, and P. Dong, "BlueVisor: A Scalable Real-Time Hardware Hypervisor for Many-Core Embedded Systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2018, pp. 75–84.

[16] S. Pinto, H. Araujo, D. Oliveira, J. Martins, and A. Tavares, "Virtualization on TrustZone-Enabled Microcontrollers? Voilà!" in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2019, pp. 293–304, iSSN: 1545-3421.

[17] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards real-time hypervisor scheduling in Xen," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, Oct. 2011, pp. 39–48.