Basswood

BALSA in Real-time

Army FACE™ TIM Paper by:

Tyler Smith[*], Dr. Rob Edman[*], and Joe Seibel[+]

Adventium Labs[*]

Carnegie Mellon Software Engineering Institute[+]

September, 2018

This material is based upon work supported by the U.S. Army Research Development and Engineering Command (RDECOM), Aviation Missile Research Development and Engineering Center (AMRDEC), Aviation Development Directorate (ADD) under contract no. W911W6-17-D-0003. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Army RDECOM or AMRDEC.

Executive Summary

Real-time performance is a critical aspect of avionics computing. The Basic Avionics Lightweight Source Archetype (BALSA) exemplar provides a collection of Units of Conformance (UoCs) backed by a Future Airborne Capability Environment™ (FACE) Unit of Portability (UoP) Supplied Model (USM) running in a Linux desktop environment. This gives an easy-to-run example for users of the FACE Technical Standard and effectively illustrates the conformance aspects of the FACE Technical Standard, but is not intended to run with hard real-time constraints. To address this limitation, we developed *Basswood*, a BALSA-based exemplar using components aligned to the FACE Technical Standard running in a real-time environment. Basswood runs on Real-Time Executive for Multiprocessor Systems (RTEMS), an open source Real-time Operating System (RTOS). Further, Basswood facilitates a practical demonstration of model-based systems engineering using the Architecture Analysis and Design Language (AADL). Basswood helps demonstrate how combined use of the FACE Technical Standard and AADL allows application of virtual integration analysis methods to FACE UoCs. This paper describes the lessons we learned adapting BALSA to a real-time environment and introduces readers to virtual integration analysis with the FACE Technical Standard and AADL.

Table of Contents

Introduction

Hard real-time timing requirements are commonplace in avionics. Indeed, the ARINC653 Operating System (OS) profile supported by the FACE Technical Standard is designed specifically to provide deterministic timing properties using time partitioning.

The BALSA exemplar has become a touchstone in the FACE community, serving as a means for introducing new members to FACE concepts and as a reference point for discussion. This paper assumes reader familiarity with BALSA. BALSA demonstrates the separation of concerns and data model driven design processes that are central to the FACE Technical Standard. However, BALSA was developed to run in a Linux environment. The Linux kernel does not provide real-time scheduling, increasing the effort required for users who wish to build upon BALSA towards airworthy FACE UoCs.

Basswood's design is derived from BALSA and will compliment BALSA in the FACE community. Basswood will be available at no cost to FACE Consortium members through the CAMET[1] library. This paper describes the design and application of Basswood, focusing on our objectives of demonstrating real-time behavior, exercising the C language bindings, and providing tools and examples with a low cost of entry.

Basswood Objective

The objective of Basswood is to accentuate BALSA; Basswood is primarily a *teaching tool*. Basswood will provide a reference point for FACE stakeholders wishing to explore real-time applications of the FACE Technical Standard.

Adventium Labs is a proponent of both the FACE Technical Standard and Architecture Analysis and Design Language (AADL). We develop tools that work with both standards and are contributing Basswood to bolster community awareness of issues surrounding real-time scheduling

Motivating Example

In BALSA the dispatch rates of the UoC threads are managed via calls to sleep and a nondeterministic Linux scheduler. Consider a scenario in which a hard real-time requirement demands that the Air Traffic Controller UoC respond to a configuration change within 100 milliseconds (ms).

The FACE Technical Standard is not intended to describe such a requirement, nor is BALSA intended to implement it. However, many safety-critical software systems must address such requirements. Basswood demonstrates that the FACE Technical Standard supports the implementation of real-time applications.

[1] camet-library.com

Architecture

Basswood is a subset of BALSA, consisting of one Portable Components Segment (PCS) UoC, one or two Platform Specific Services Segment (PSSS) UoCs, and a Transport Services Segment (TSS) library connecting them (see Figure 1). We adapted the C++ source code for the BALSA Air Traffic Controller (ATC) and Embedded Global Positioning System (GPS)/Inertial Navigation System (INS) (EGI) to C source code and replaced the User Datagram Protocol (UDP)-based TSS with one using POSIX message queues. In some examples, we also use an AirConfig PSSS UoC. Each PCS and PSSS UoC is implemented as a thread (RTEMS does not provide memory space separation).

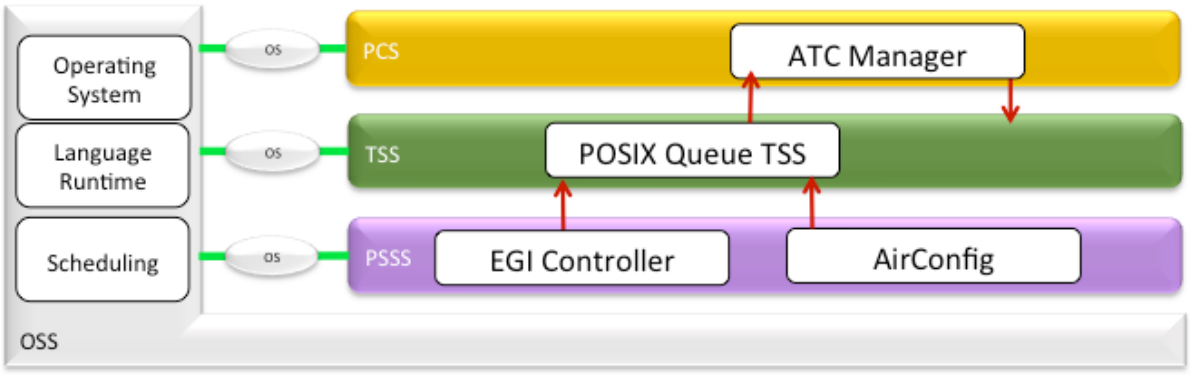

Figure 1 Basswood Architecture, Diagram Derived from A BALSA Integration and Test Session (The Open Group, 2016)

FACE Technical Standard Version

The Basswood USM adheres to the FACE Technical Standard, Edition 3.0 metamodel (The Open Group, 2017).

Due to the limited availability of FACE 3.0 data modeling and conformance checking tools, we implemented Basswood using the FACE Technical Standard, Edition 2.1 code generation tools. Future versions of Basswood will be updated to use tools based on the FACE Technical Standard, Edition 3.0.

C Language Bindings

We used the BALSA 2.1 USM and Vanderbilt University's Generic Modeling Environment[2] (GME) FACE tools to generate C source code for the Basswood TSS.

Operating System

We used Real-Time Executive for Multiprocessor Systems[3] (RTEMS) version 5 as the demonstration platform for Basswood. We used the RTEMS Rate Monotonic Scheduler (RMS) to set deterministic timing properties for the Basswood threads.

[2] http://www.isis.vanderbilt.edu/Projects/gme/

[3] https://www.rtems.org/

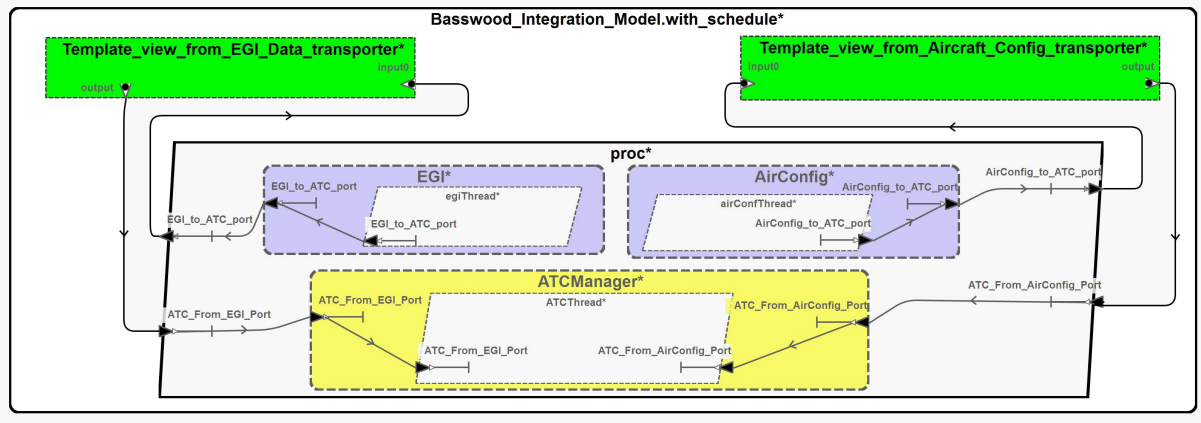

Figure 2 Basswood Threading and Message Passing

We run Basswood in the QEMU (Quick Emulator) virtualization environment. Execution of Basswood on a bare-metal device running RTEMS is possible, but for practical purposes execution in QEMU is sufficient. Although QEMU does not guarantee wall-clock relative hard real-time performance, it *does* provide deterministic timing characteristics for events occurring inside the scope of a single QEMU instance.

Figure 2 shows a graphical AADL model of Basswood, color-coded in the same manner as Figure 1 to represent association to particular FACE segments. There is a single process (memory space) called proc, which contains three threads groups: EGI, AirConfig, and ATCManager. Each thread group corresponds to a single UoC. Each UoC has a single thread. The TSS layer is abstracted as two *transporters* per the FACE Integration Model.

The FACE Technical Standard and AADL

What is AADL?

The Architecture and Analysis Design Language (AADL) is a Society of Automotive Engineers (SAE) Aerospace Standard (AS) system model specification language (AS5506C) that supports various types of performance and safety analysis. AADL is a semantically precise modeling language for describing aspects of cyber-physical systems using standardized textual and graphical representations. AADL focuses on model-based analysis of static and dynamic embedded computing system properties. AADL supports modeling at various levels of detail through delayed specification and model refinements and extensions. For example, a high-level system model might have an abstract Transport Services Segment (TSS) library that is refined in subsequent models to a Common Object Request Broker Architecture (CORBA)-based TSS library or to a Transport Control Protocol /Internet Protocol (TCP/IP)-based TSS library. There are analyses for many performance and quality metrics and there are tools to help integrate systems. For example, AADL analyses allow you to evaluate the costs and benefits to size weight and power (SWaP), safety, security, timing, scheduling, resource utilization, and performance of different refinements (SAE International, 2017).

Why AADL?

The FACE Technical Standard and AADL are complementary standards. AADL provides tools for modeling and analyzing aspects of cyber-physical systems that are outside of the scope of the FACE Technical Standard. AADL adds hardware representations and hierarchical system compositions to the rich software data structures captured in the FACE Technical Standard, providing additional capabilities for detection and prevention of hardware implementation errors. Using the FACE Technical Standard and AADL together provides for early detection of many classes of integration errors.

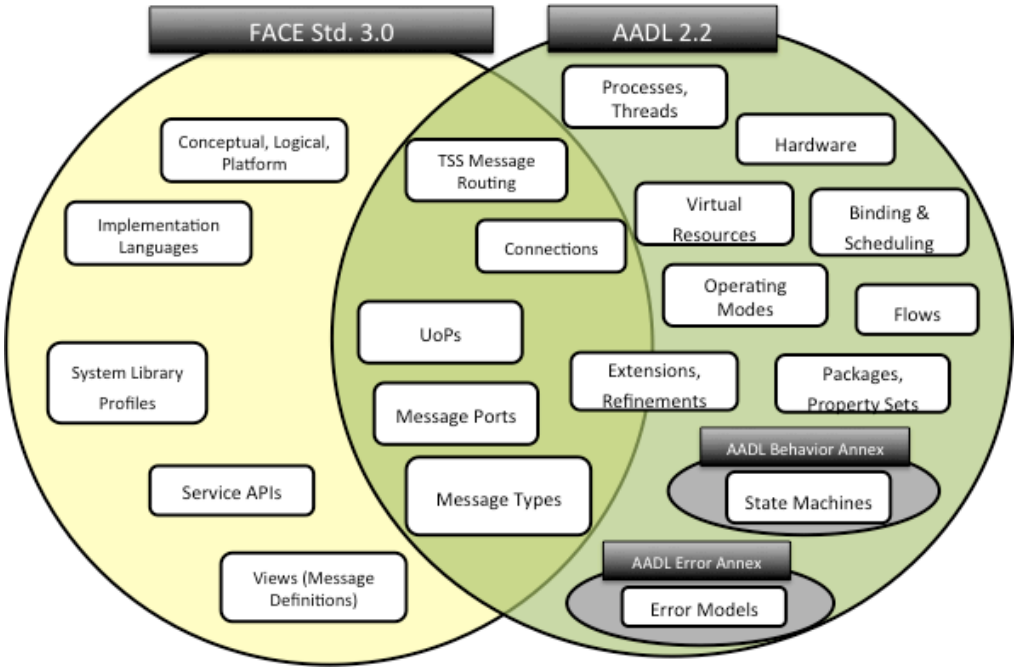

Figure 3 Comparison of AADL and the FACE Technical Standard

How does AADL Relate to the FACE Technical Standard?

The FACE Technical Standard provides a means to describe UoCs and the data they exchange, whereas AADL provides the capability to model a cyber-physical system constructed from FACE UoCs and extended with hardware and operational properties. AADL supports modeling software structures such as data types and threads, hardware bindings, and environmental context to facilitate analysis of the interactions between system components in both the hardware and software domains. AADL and the FACE Technical Standard overlap in their capacity to describe some system features but focus on different system characteristics and on highlighting different classes of errors (see Figure 3). AADL is particularly good at detecting errors that only manifest in integrated systems. For example, AADL analyses can detect conditions like total memory exceeded, fully loaded processors fail to meet all deadlines, or unhandled error propagation flows.

The AADL Annex for the FACE Technical Standard

The AADL Annex for the FACE Technical Standard[4] provides guidance for translating a FACE Technical Standard Edition 3.0 Data Architecture eXtensible Markup Language (XML) Metadata Interchange (XMI) model into AADL so that behavior and timing properties can be added and analyzed. The annex supports the modeling, analysis, and integration of FACE artifacts in AADL. It gives AADL style guidelines and an AADL property set to provide a common approach to using AADL to express architectures that include FACE components. Using common properties and component representations in AADL makes AADL models of FACE components portable and reusable and increases the utility of tools that operate on such AADL models. The annex will be submitted for inclusion in the official SAE standard AADL.

Using AADL with Basswood

We used *the FACE Data Model to AADL Translator*[5] to translate the Basswood USM to AADL following the conventions of the AADL Annex for the FACE Technical Standard.

The *FACE Data Model to AADL Translator*

The *FACE Data Model to AADL Translator* converts a `.face` file aligned with the FACE Technical Standard, Edition 3.0 into an AADL 2.2 model (The Carnegie Mellon Software Engineering Institute, 2018). For example, the translator produces AADL `data` classifiers for the conceptual, logical, and platform entities, `thread groups` for PSSS and PCS UoCs, and a `system` component for an integration model. The system contains `subcomponents` for transport nodes and `connections` between these subcomponents.

The purpose of the translator is to enable developers of FACE UoCs to utilize the analytical capabilities of AADL. The architectural information included in the FACE USM is captured and expressed in the resulting AADL model. This model can then be extended by the user to add various properties and then analyzed. The *FACE Data Model to AADL Translator* allows for changes in a FACE USM to be quickly realized and analyzed in an AADL model.

The *FACE Data Model to AADL Translator* is a set of plugins to the Open Source AADL Tool Environment (OSATE). The translator integrates with the OSATE modeling environment and is invoked from within OSATE.

Using the Translator with Basswood

[4] The annex is available at https://www.adventiumlabs.com/sites/adventiumlabs.com/files/2018.04.09.2_annex_for_face_ed_3_0.pdf

[5] See http://osate.org/additional-components.html for more information on the *FACE Data Model to AADL Translator*

To create the AADL model used to analyze Basswood, we used the *FACE Data Model to AADL Translator* to convert the `.face` XMI USM file into AADL packages. The translator creates up to four packages, depending on the level of detail provided in the model. In our case, the FACE USM contained PSSS and PCS UoCs and an integration model, so the translator created four AADL packages.

Motivating Example Revisited

In the introduction we described a scenario in which the Air Traffic Controller (ATC) UoC was required to respond to a configuration change within 100ms. Basswood enables us to reliably demonstrate this scenario.

Consider a Basswood configuration in which each of the ATC, EGI, and AirConfig UoCs is assigned a priority and a fixed rate dispatch schedule according to a Rate Monotonic Schedule (RMS). ATC runs at a high rate and is assigned a high priority. EGI runs at a medium rate and is assigned a medium priority. AirConfig runs at a low rate and is assigned a low priority. Figure 4 shows a graphical representation of this scenario.

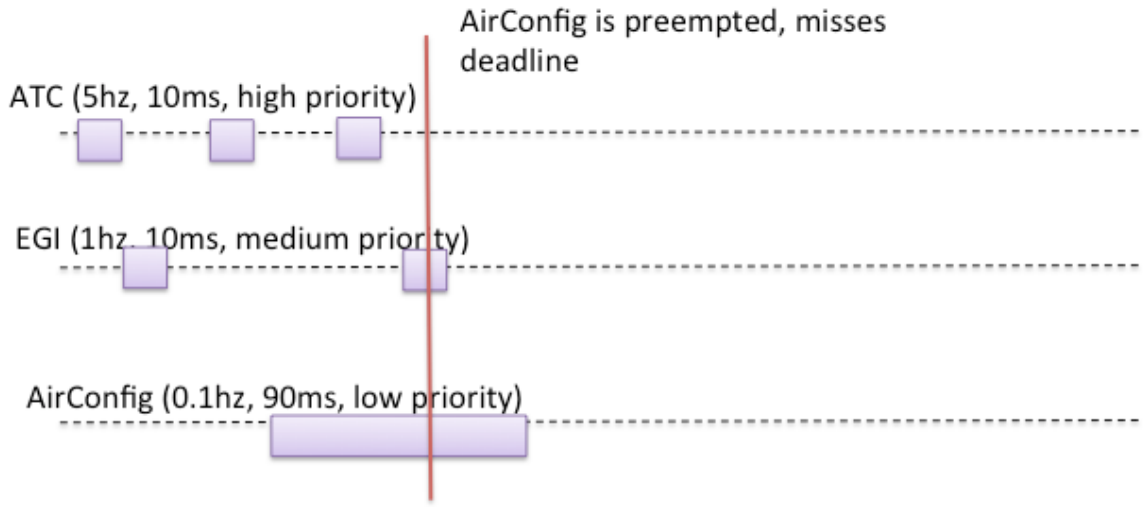

Figure 4 Example Priority Preemption Scenario

As a low priority thread, the AirConfig UoC is vulnerable to preemption by higher priority threads. This means that although AirConfig's Worst Case Execution Time (WCET) is only 90ms, if the EGI is dispatched during AirConfig's execution then AirConfig may be preempted and may miss the deadline for sending updated configuration to the AirTraffic Controller.

Basswood's real-time determinism and the real-time scheduler in RTEMS allow us to demonstrate this scenario. Similarly, AADL's timing semantics allow us to configure and analyze this scenario in AADL. Together these models provide a robust training platform.

Lessons Learned

The C++ BALSA Build is Difficult to Port

We originally intended to use the consortium-released BALSA 2.1 source code (written in C++) in RTEMS. Several complications caused us to switch our plan to a subset of the BALSA architecture and to change our implementation language to C. First, the BALSA architecture uses a UDP-based TSS. Although not complicated to implement, most (open source) IP networking stacks are not intended for hard real-time application, and we feared that UDP would prevent reliable demonstration of performance problems. Second, we encountered compilation problems with the Makefiles in the BALSA system and the need to render them compatible with the RTEMS build environment. BALSA assumes a standard Linux build environment, and uses hierarchy of Makefiles. Unfortunately, many of them seemed to require changes for an RTEMS build. Some variables pass from one Makefile to another, but others are reset in subordinate Makefiles. We also found some libraries could not be resolved: `Librt` seemed to be missing; certain `pthread` functions seem to be missing (at least in release version of RTEMS).

We also had trouble getting the RTEMS-provided C++ test programs to run correctly, specifically ones involving exceptions and ones involving iostreams (iostreams are not permitted, per the FACE Technical Standard). The same behavior was observed with our own test programs: attempts to exercise either feature caused runtime hangs. Those are not major impediments with BALSA, but it raised a question about what other C++ runtime issues we might need to code around.

Recommendations to the BALSA Developers: Consider switching to an automake[6] configuration to ease issues surrounding Makefiles.

Sharing Rules for BALSA are Complex

The BALSA exemplar has varying releasability across versions. One variant of BALSA 2.0 was approved for public release, but BALSA 3.0 is not yet approved. We are pursuing consortium clarification on releasability of derived artifacts before sharing our Basswood USM, as it is derived from BALSA models.

Recommendations to the Integration Workshop Standing Committee: Post specific guidance regarding the release of BALSA-derived work.

[6] https://www.gnu.org/software/automake/

References

SAE International. (2017, January). *Architecture Analysis & Design Language (AADL) Aerospace Standard (AS) 5506 Revision C.* From https://www.sae.org/standards/content/as5506c/

The Carnegie Mellon Software Engineering Institute. (2018, June). *FACE Data Model to AADL Translator.* Retrieved July 31, 2018 from OSATE: http://osate.org/additional-components.html

The Open Group. (2017, November 15). *FACE Standard 3.0.* Retrieved 2018 from The Open Group: https://publications.opengroup.org/c17c

The Open Group. (2016, September 27). Pilot BITS Event. *Future Airborne Capability Environment* . The Open Group.

About the Authors

Tyler Smith – Adventium Labs

Mr. Smith is a senior research scientist at Adventium Labs and is leading Adventium's work supporting the Army Joint Multi-Role (JMR) Mission Systems Architecture Demonstration (MSAD) effort in model based procurement and continuous virtual integration. Mr. Smith is responsible for the design and implementation of the Continuous Virtual Integration Server (CVIS) that enables users to stand up a server to automatically execute scripts for model retrieval, integration, analysis, and report generation. Likewise Mr. Smith is coordinating an Architecture Centric Virtual Integration Process (ACVIP)-aligned exercise to evaluate AADL model based procurement and integration with Single Source of Truth (SSoT) practices and tools. In addition, Mr. Smith was the Principal Investigator (PI) on a Phase I SBIR (Small Business Innovative Research) building tools for integration of virtual and physical components in a lab setting for early testing. Finally, Mr. Smith is a co-author of the AADL Annex for the FACE Technical Standard, version 3.0. The Annex was released in February 2018 and has been distributed to both the AADL and FACE Technical Standard communities. Prior to joining Adventium in 2014 Mr. Smith was a software engineer with General Dynamics and worked on components aligned to the FACE Technical Standard Edition 2.0. Mr. Smith is a member of the AADL Standards Committee and is Adventium's primary representative in the FACE Consortium.

Dr. Rob Edman – Adventium Labs

Dr. Edman is a senior research scientist at Adventium Labs and has participated in a range of projects involving real-time scheduling, embedded systems, and computer networking. He has served as PI on multiple Phase SBIRS, including Adventium's current Phase II SBIR on virtual integration of behavioral aspects of components aligned to the FACE Technical Standard. Edman has been deeply involved in generating analyzable models for real-time scheduling problems and creating solvers for those models using SMT tools. These models included process allocation, memory constraints, timing analysis of threads, bandwidth limited communications, and asynchronous timing boundaries. The prototype SMT tools generated from Dr. Edman's work formed the basis for the schedulability analysis tools available through CAMET. On the Navy-funded Mixed Criticality, Assured, Real-Time VMM TM (MiCART) program, Dr. Edman demonstrated that critical and non-critical software processes could run side by side on the same virtualization base. As part of this effort, he helped implement a static scheduler in Xen to provide performance guarantees on multi-core systems and created AADL models capturing the performance requirements of real-time software running within the system. Dr. Edman is a co-author of the AADL Annex for the FACE Technical Standard, version 3.0. Dr. Edman is a member of the AADL Standards Committee and is one of Adventium's representatives in the FACE Consortium.

Joe Seibel – Carnegie Mellon Software Engineering Institute

Mr. Seibel is a Member of the Technical Staff at the Software Engineering Institute (SEI). He is one of the main developers of OSATE. His focus is on language implementation using the Xtext framework. He also works on OSATE's UI and recently developed the *FACE Data Model to AADL Translator*. Mr. Seibel is a member of the AADL Standards Committee and is one of the SEI's representatives to the FACE Consortium.

About The Open Group FACE Consortium

The Open Group Future Airborne Capability Environment (FACE) Consortium was formed as a government and industry partnership to define an open avionics environment for all military airborne platform types. Today, it is an aviation-focused professional group made up of industry suppliers, customers, academia, and users. The FACE Consortium provides a vendor-neutral forum for industry and government to work together to develop and consolidate the open standards, best practices, guidance documents, and business strategy necessary for acquisition of affordable software systems that promote innovation and rapid integration of portable capabilities across global defense programs.

Further information on FACE Consortium is available at www.opengroup.org/face.

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. Our diverse membership of more than 600 organizations includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The Open Group aims to:

Capture, understand, and address current and emerging requirements, and establish policies and share best practices

Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies

Offer a comprehensive set of services to enhance the operational efficiency of consortia

Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.