Reusable Transport Services (TS) Software for Instantiations of the Future Airborne Capability Environment (FACE™,[1]) Architecture

NAVAIR FACE™ TIM Paper by:

Trevor Stittleburg

Georgia Tech Research Institute

September, 2017

[1] FACE™ and logo design are trademarks of The Open Group in the United States and other countries

Table of Contents

Executive Summary

The Future Airborne Capability Environment (FACE™) Technical Standard defines a "Reference Architecture intended for the development of portable software components targeted for general-purpose, safety, and/or security purposes" (Reference 1). Within this reference architecture, software which implements the Transport Services (TS) interfaces performs the tasks of transferring data between applications (e.g., relaying requests and responses). This software resides within the Transport Services Segment (TSS), a logical construct in the FACE Reference Architecture. The TSS is a critical part of the architecture for integrators (as well as developers).

This paper describes an implementation of the Transport Services Segment capabilities developed by Georgia Tech Research Institute (GTRI) under contract with the US Navy (NAVAIR/PMA 209) and US Army (AMRDEC/PEO Aviation). The Reusable Transport Service (Reusable TS) is a set of software in alignment with the TSS requirements and provides the core transport capabilities such as message distribution and routing, serialization, and data transformation. The Reusable TS is portable and configurable so that it can be used in many systems employing the FACE architecture. The software is distributed with documentation and templates aimed at integrators who need a solution for a system employing the FACE architecture. It is developed in C++ and is aligned with the FACE Technical Standard, Edition 2.1. Extensions and upgrades can be developed in the future to support more transport types, functionality, and future versions of the FACE Technical Standard.

GTRI has used the Reusable TS in demonstrations and the software is now being used in other projects which use the FACE Technical Standard. Reuse of TSS software eases integration and allows more focus on other work products such as components in the other segments of the reference architecture.

Background

Transferring data between applications is often performed by a software layer dedicated to this activity. This software serves two purposes: (1) it abstracts these tasks from applications to improve separation of concerns, (2) it allows software components in different environments to interact (for example, components that differ by process, programming language, operating system partition, location on a network, etc.).

Open architectures employ standardized interfaces, behaviors, and other aspects to enable reuse of components and simplify integration. Standardization of interfaces is key to the openness of software, and standardizing the interface to this software layer is one way open architectures achieve their goals of reuse, portability, and increased competition among implementers. Furthermore, reuse of the data transport layer itself is usually a goal of the architecture and the implementer. Data transport functionality is typically generic and reusable on a range of systems that use the architecture, and may support several forms of inter-process communication (IPC).

The Future Airborne Capability Environment (FACE™) Technical Standard is an open systems architecture which defines a layered software reference architecture for avionics that utilizes well-defined open interface (Reference 1, hereafter referred to as the "FACE Technical Standard"). The reference architecture is divided into segments, and interfaces between segments are standardized. A diagram of the FACE Reference Architecture segments is shown in Figure 1. Communication between applications in the Portable Component Segment (PCS) and Platform Specific Services Segment (PSSS) is performed by using the Transport Services interface to access transport-related functionality in the Transport Services Segment (TSS).

A key goal of the FACE Technical Standard is to promote reusability of components. To actually use components in the architecture, the TSS must be provided so that they can communicate. This can be implemented in a number of ways, and can potentially be implemented in such a way that it is highly specific to a particular system with little reusability.

If the implementation of the TSS capabilities is tightly coupled to the system, then modification of the applications in the system may require modification of the TSS, and the TSS itself may not be reusable on other platforms. In this type of scenario, the implementer of the TSS could exert control over the integration and make future changes or upgrades more difficult. For example, if the TSS implementation is not sufficiently configurable, then it may have to be recompiled and relinked to make small changes like changing connection names, port numbers, addresses, etc. Furthermore, if the TSS is not reusable, the same functionality might have to be re-developed for another system, increasing cost and development time.

On the other hand, if the TSS is developed to maximize reuse, configurability, and portability, then small changes could be made without even recompiling the source code (e.g. though use of configuration files, separation of type-specific code into separate objects), and significant parts of the functionality could be easily re-used on a number of platforms.. This paper describes a reusable, portable implementation of the TSS functionality which enables reusability, reduces integration efforts, and helps prevent vendor lock.

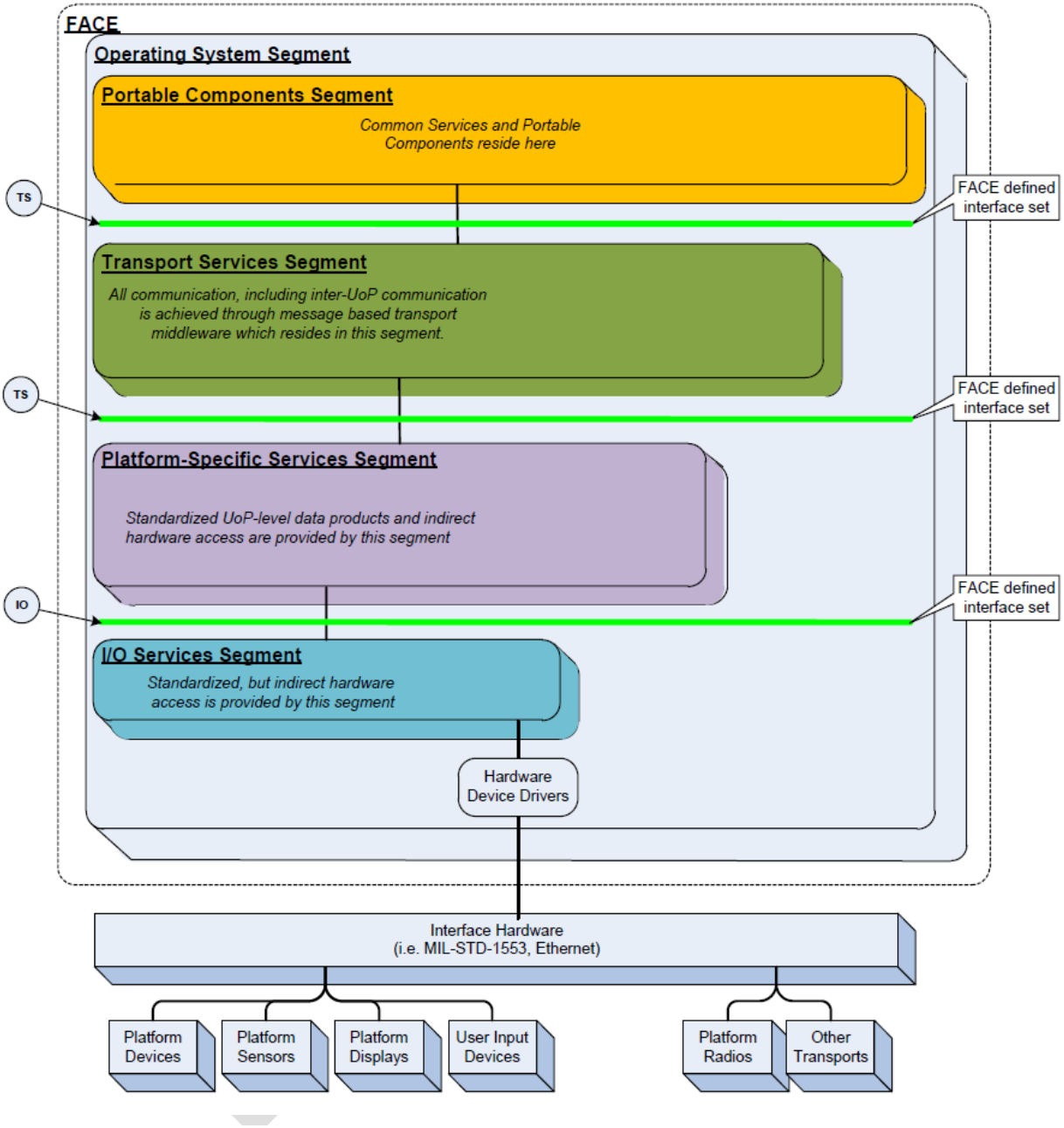

Figure 1. FACE Reference Architecture (Source: Reference 1)

Reusable Transport Service (TS)

The Reusable Transport Service (TS) is a set of software packages, documentation and artifacts developed with the goal of implementing reusable, portable TSS functionality and satisfying the requirements of the FACE Technical Standard. The Reusable TS is not a service in the sense of a process in a service-oriented architecture, but rather takes its name from the TSS. The software was developed by Georgia Tech Research Institute (GTRI) under contract with the US Navy (NAVAIR) and US Army (AMRDEC, PEO Aviation).

The high-level goals of the Reusable TS are:

- Implement the TS (Transport Services) interface
- Implement the Type Abstraction (TA) interface as a Unit of Portability (UoP)
- Provide the following TSS capabilities defined in the FACE Technical Standard: Transport Service capability, Distribution capability, Configuration capability, and Data Transformation capability
- Enable reusability of the TS for arbitrary systems using the FACE Reference Architecture (to the extent possible)

The Type Abstraction (TA) interface is defined in the FACE Technical Standard as an internal interface of the TSS. The TS interface uses the TA interface to fulfill type-independent functions like the distribution of serialized data along transport mediums such as inter-process communication (IPC) or a network. The implementation of the TA interface can be a stand-alone UoP. The Reusable TS implements both the TS and TA interfaces. The TS interface implementation depends on the TA UoP for distribution functions. For the purposes of this paper, the "Reusable TS" will refer to both the TS implementation and the Reusable TA UoP (unless otherwise specified).

Part of implementing the TS interface involves implementing strongly typed functions defined in the interface itself and possibly other strongly typed functions internal to the TSS where applicable (transformations, serialization). The types used by a given system are specific to that system. The types are documented in the Unit-of-Portability Supplied Model (USM), a data model provided by implementers of FACE Units of Conformance (UoCs), and the language-specific definitions are defined by the language mappings in the FACE Technical Standard. The types are therefore specific to the components in the system.

In order to achieve a reusable TS implementation, the typed portion of the functionality is left for the integrator to complete, and all interfaces to the typed portion are defined in the Reusable TS documentation (References 2 through 5) and the FACE Technical Standard (Reference 1). These typed interfaces and definitions are readily generated from the USM. A tool (Corinth) for generating serialization and deserialization functions is provided, but the development of tools to generate the other definitions is left to integrators. This tool is described in Reference 7.

A depiction of a complete Transport Services Segment that makes use of the Reusable TS is shown in Figure 2.

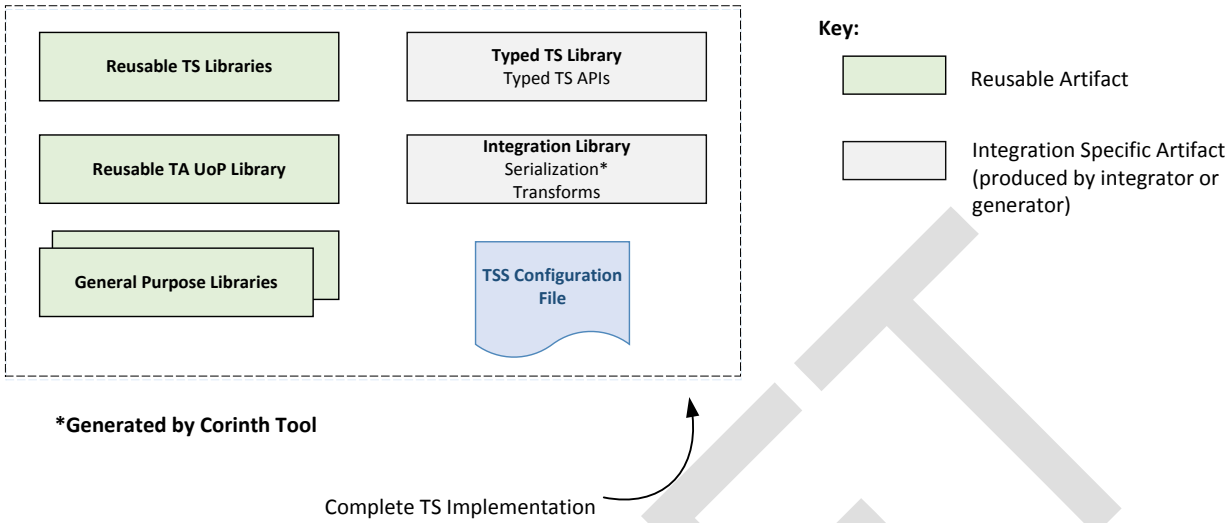

Figure 3. Depiction of Complete TSS Using Reusable TS

Applications in the PSSS or PCS use the TS interface directly. The TS interface uses the Type Abstraction interface internally to implement some of the type-independent functionality of the TSS, as shown in Figure 3.

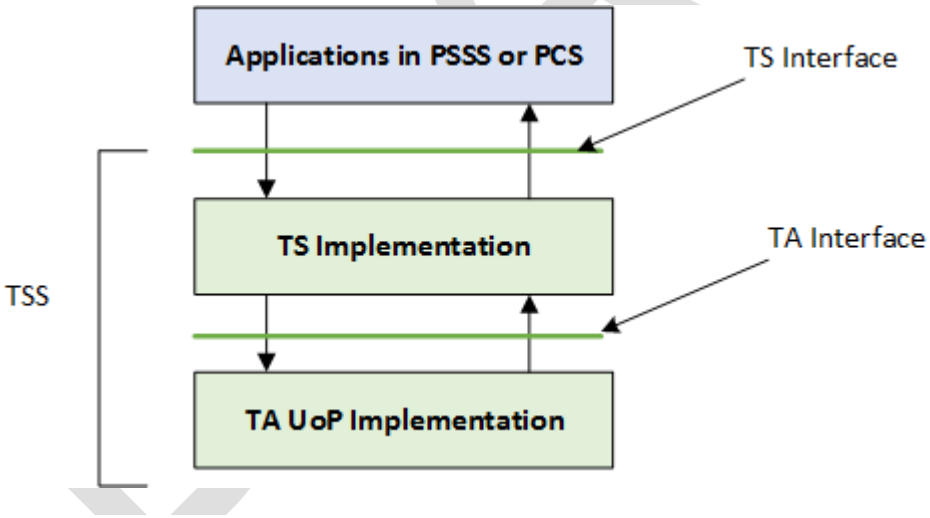

Figure 2. Relationship of TS Interfaces

Reusability was achieved by implementing the software aligned with the Safety Base profile of the FACE Technical Standard, tested on multiple operating systems (Lynx Software Technologies LynxOS, GreenHills INTEGRITY, Red Hat Enterprise Linux (RHEL)), and providing configurability of most transport functions (e.g. connection information, networking or inter-process communication used). Additionally, to enable interoperability between FACE UoCs on different systems, basic data serialization/deserialization was implemented.

Example Integrator Use

As discussed in the previous section, the Reusable TS can be used by a system integrator to provide capabilities of the TSS. However, the integrator must supply some integration-specific functions in a defined format. These functions are specific to the types used by the PCS and PSSS components the integrator is using (or implements, if they also implement the components). This code is then compiled and linked along with the Reusable TS libraries and headers to form a complete TSS for the system. This process is shown in Figure 4 and Figure 5.

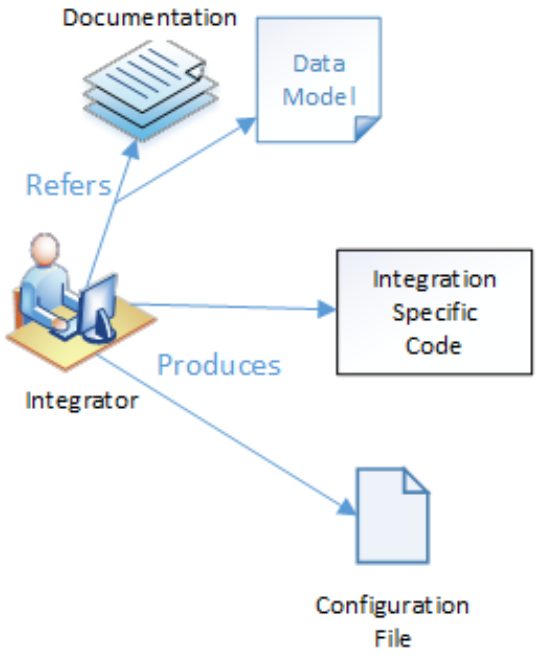

Figure 4. Integration Process – Producing Integration Specific Artifacts

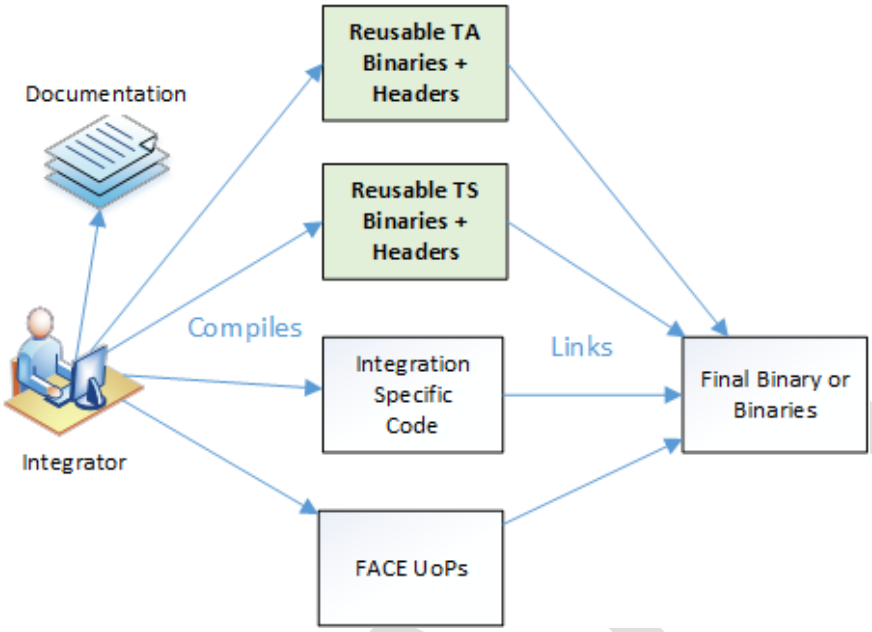

Figure 5. Compiling and Linking Reusable TS, Integration Specific Code, and UoP

Alternatively, if using the source distribution of the Reusable TS software, the integrator may compile the Reusable TS software from source.

The strongly typed, integration specific code must implement the functions defined by the documentation and templates provided with the Reusable TS. These functions include simple tasks such as:

- Type Definition Globally Unique Identifier (GUID) lookup
- Type name as a string lookup
- Calling type-specific construction of objects

Serialization and deserialization functions are generated by a tool (Corinth) with the Reusable TS distribution which uses the FACE User Supplied Model (USM) for the system. The FACE USM is part of the development of a Transport Services Segment for a system and is specific to the system. This must be produced by the developer/integrator using the Reusable TS as required by the FACE Technical Standard. Datatype source code must be produced by using the Datatype Generator tool distributed by The Open Group as part of the Modeling Tools for FACE Software Development distributed by Vanderbilt University (Reference 6).

Design and Operation

The Reusable TS implements the configuration, distribution, and transformation capabilities of the TS interface as a set of libraries to be integrated with a FACE Unit of Conformance (UoC). The Reusable TS exposes the following concepts to the integrator via configuration information. All the concepts below are configured in a file written in eXtensible Markup Language (XML), and can therefore be changed without re-

compiling the source code (Some exceptions apply such as when Data Distribution Service (DDS) libraries are used – see next paragraph). The file also contains the configuration for the Reusable TA UoP in a subsection.

A channel is a system resource used to transport data. This is part of the Reusable TA UoP configuration. In current version of the Reusable TS and TA UoP, there are three supported types of channels: ARINC653 queuing ports, POSIX message queues, and UDP sockets. Note that a channel is defined separately from a connection. DDS using OpenDDS is also supported, but in this build of the Reusable TS, it is not aligned with nor conformant to the FACE Technical Standard's POSIX API profile limitations (due to the fact that OpenDDS is not aligned with the FACE Technical Standard).

A connection is a UoP's connection to the TSS as described in the TS API in the FACE Technical Standard. A UoP in the PCS or PSSS uses one or more connections to send and receive messages via the TSS. Connections are type-specific, and for the purposes of the Reusable TS, they are also direction-specific (inbound or outbound). The TS must be initialized using the TS function Initialize() before creating a connection, and a connection must be created with Create_Connection() function before it can be used. Each connection must contain at least one subconnection.

A subconnection is a set of data representing a single source or destination UoP for a TSS connection. Because the TS interface abstracts the actual destination or source of data from the application, this internal concept is added so that these details can be configured by the integrator. A single TSS connection may send to multiple destinations, or receive from multiple sources. Each source or destination is termed a subconnection. Each subconnection must be mapped a single channel. In this version of the Reusable TS, a channel may only be mapped to one subconnection.

A serialization scheme is a method of serializing and deserializing data while it is transported across the TS. Serialization is a process of converting structured data in the application to an unambiguous, cross-platform format capable of being stored or transmitted and then reconstructed into the structured data again. Many serialization formats are character-based (e.g., XML, JSON, SOAP). Serialization in the Reusable TS is currently supported using a packed binary format. Other serialization formats could be added in future versions.

A callback thread is a thread of execution that is run in the background of a process to receive data on one or more TSS connections. These are configured and grouped together by a callback manager. Threads can be configured to allow registration of particular TSS connections and to use a particular timeout for receiving data on each connection. Typically, only one thread per TSS connection is desired. If the callback manager is not configured, the default behavior will be to create one thread every time a TSS connection is registered with Register_Read_Callback() function. Callback threads are implemented as POSIX threads.

A datatype transformation (or transform) is an optional function that converts one datatype to another datatype. Both datatypes must be represented in the FACE Unit of Portability (UoP) Supplied Model (USM) for the system. A transform sequence is an ordered sequence of one or more transforms. Each subconnection can have a transform sequence optionally mapped to it, which allows connections with different datatypes to communicate so long as a sufficient transformation is provided. The transformation functions must be defined by the developer/integrator and are referenced by name in the configuration file. For example, this allows an outbound connection with Datatype A to communicate with an inbound connection with Datatype B, given that the transform sequence transforms an instance of Datatype A to an instance of Datatype B.

At initialization (during the call to Initialize()), the Reusable TS reads the configuration file which contains the configuration of all the above elements. It creates objects and saves configuration data to prepare the TS interface to be used according to the configuration.

When a UoP makes a call to Send_Message(), the Reusable TS will attempt to send it to all subconnections configured for that TSS connection. For each subconnection, the serialization scheme's serialize method will be applied to the data before being sent.

Similarly, when a UoP calls Receive_Message(), the Reusable TS will try to receive from each of the subconnections' channel for that TSS connection. When a message is received on a subconnection, that message will be returned from Receive_Message(). The serialization scheme's deserialization method will be applied to the data before it is returned to the UoP.

A depiction of a simple Send_Message() operation which uses a single data transformation is shown in Figure 6. Note the connection contains a subconnection which references a transport channel (same as a channel as described previously). The transport channel is a resource which is defined by the Reusable TA UoP and is referenced by the Reusable TS in its configuration data. The channel name corresponds to the connection name in the Type Abstraction TS API.

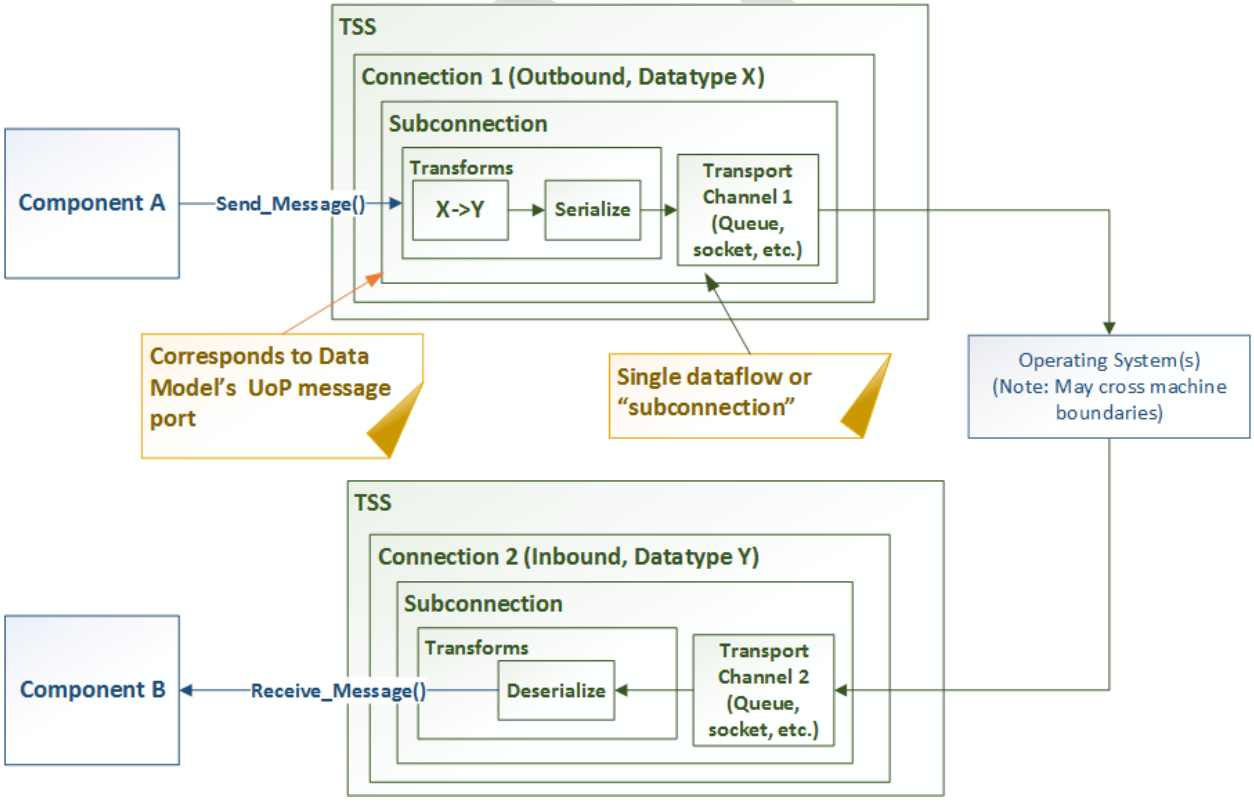

Figure 6. Depiction of Single Message Transported from Component A to Component B

A depiction of a one-to-many Send_Message() operation is shown in Figure 7.

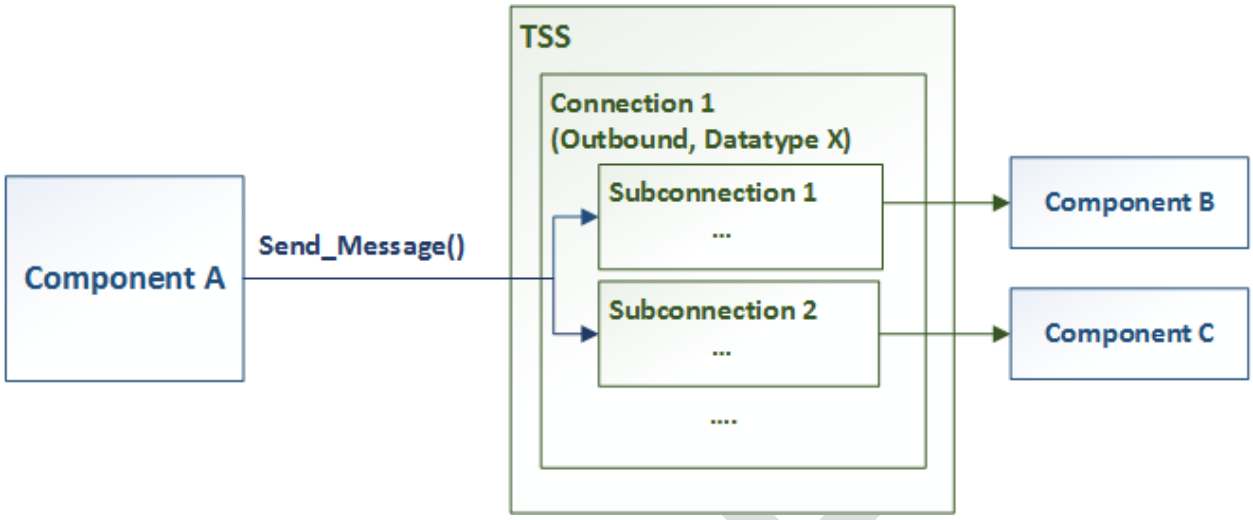

Figure 7. Depiction of One-to-Many Message Send Operation

In this depiction, each subconnection is shown to correspond to a single destination (receiving component). Other aspects of the configuration such as the transport channel are left out in this diagram.

The use of callbacks is also supported by the Reusable TS. The Register_Read_Callback() interface is implemented and uses threads to read on TSS connections and use function callbacks appropriately when a message is received (callback threads can either be defined in configuration or a default behavior will create a new thread for each connection).

Datatype transformations can be chained together through configuration (see Figure 8). The output of each transformation must match the input of the next transformation. Serialization is performed after the final transformation. This may be useful if transformations from A to B and B to C are already available but a transformation from A to C is not available.

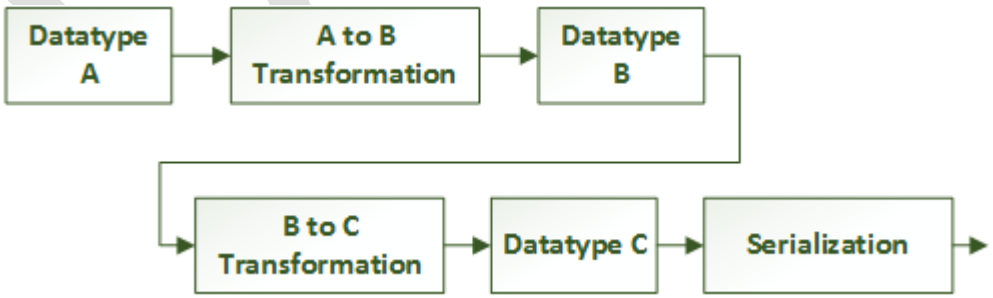

Figure 8. Multiple Datatype Transformations

Conformance Testing

At the time of this writing, the Reusable TS and Reusable TA UoP have not been evaluated by a FACE Verification Authority (VA). However, the software is aligned to the FACE Technical Standard and internal testing of the software has been performed successfully using the FACE Conformance Test Suite (CTS) and the Conformance Verification Matrix (CVM).

As discussed in previous sections, a complete TSS, which includes the integration-specific, strongly typed function implementations, can undergo verification to meet the requirements of the FACE Technical Standard. However, for a given system, the implementation of the Transport Service (TS) interface must be tested and undergo the verification process, which includes the type specific functions. The Reusable TS (except the TA UoP) cannot be independently tested outside of an integration because of the type-specific interfaces in the TS interface. However, because the Reusable TS libraries are aligned to the FACE Technical Standard, most of the TSS code will already be aligned with the requirements of the FACE Technical Standard.

To exercise the functionality of the Reusable TS and to allow for unofficial, internal conformance testing, a mock system was developed which uses the Reusable TS software. This Reusable TS System Integration (RTSI) provides a complete TSS which can be run through the FACE Conformance Test Suite (CTS).

In the Figure 9, a simplified diagram of the RTSI is shown. The Reusable TS has been tested internally using the CTS using the following types of tests with the Safety Base profile:

- TS using Type Abstraction interface (using the RTSI TSS which encompasses the Reusable TS)
- Type Abstraction UoP (the Reusable TA UoP is tested as a Type Abstraction UoP)

Additionally, the FACE Conformance Verification Matrix (CVM) has been used to internally ensure the Reusable TS satisfies all applicable requirements.

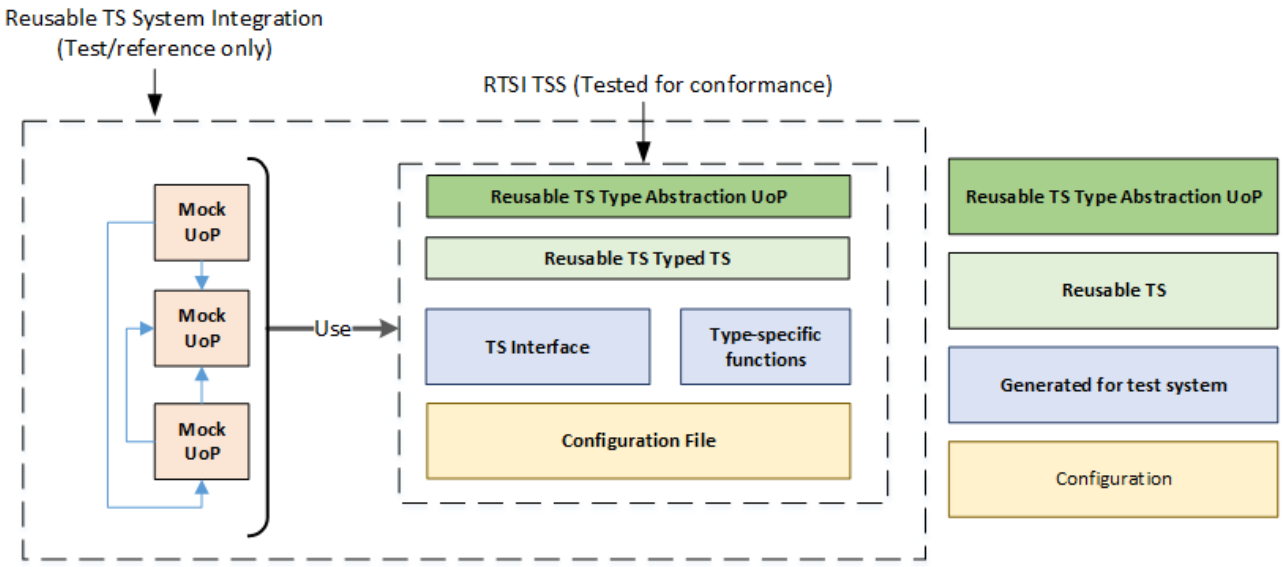

Figure 9. RTSI Architecture

Conclusions and Future Work

The TSS is perhaps the most critical part of an integration effort of components in instantiations of the FACE Reference Architecture. The interfaces are well defined by the FACE Technical Standard, but the implementation is left to the developer. A reusable, portable implementation of the TS interface allows integration efforts to avoid re-implementation of TSS functionality. Additionally, incremental improvements to the implementation can be made and carried forward to future versions. The Reusable TS is a configurable, reusable, and portable implementation of the TS and TA interfaces which fulfills the core functions of a TSS in a FACE Reference Architecture. It supports several transport types as well as binary serialization, and these can be extended in the future.

When future versions of the FACE Technical Standard are released, it is anticipated much of the functionality of the Reusable TS software will be able to be reused and a version could be produced that aligns with the requirements and interfaces of the new standard(s).

Additional transport mechanisms (such as shared memory or sampling ports) can be added to the Reusable TS in future versions. Although these cannot be directly added by an integrator, the internal interfaces are such that addition of new transport mechanisms is a simple process. Security measures such as encryption can also be introduced either in the TA UoP.

References

(Please note that the links below are good at the time of writing but cannot be guaranteed for the future.)

1. Technical Standard for the Future Airborne Capability Environment (FACE™), Edition 2.1, © 2014 The Open Group. FACE™ is a trademark of The Open Group in the United States and other countries.
2. Georgia Tech Research Institute (GTRI), Reusable TS Software Design Description, Rev. F, December 2016
3. Georgia Tech Research Institute (GTRI), Reusable TS Software Version Description (Source), Rev. F, December 2016
4. Georgia Tech Research Institute (GTRI), Reusable TS Integration Guide, Rev. D., December 2016
5. Georgia Tech Research Institute (GTRI), Reusable Type Abstraction UoP Software Version Description (Source), Rev. C, December 2016
6. FACE Downloads, Institute for Software Integrated Systems (Vanderbilt University), http://www.isis.vanderbilt.edu/FACE.
7. Georgia Tech Research Institute (GTRI), Corinth Code Generators, Software Version Description, Rev. C, December 2016

About the Author(s)

Trevor Stittleburg is a software engineer with the Electronic Systems Laboratory (ELSYS) at Georgia Tech Research Institute (GTRI). As the lead engineer on the Reusable TS project, he worked on designing an implementation of the TSS requirements in a portable way and provided feedback on various aspects of the TSS interfaces to NAVAIR, a member of the FACE Consortium.

GTRI has been involved in the development of the FACE Technical Standard by supporting NAVAIR and Army (AMRDEC and PEO Aviation) since the early stages of the program. GTRI continues to support the government in improving and facilitating adoption of the standard.

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 IT standards. With more than 500 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – 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.