Reusable Transport Services (TS) Software

NAVAIR FACE TIM
October 17, 2017

PMA209 develops, integrates, and delivers avionics solutions that meet customer requirements, enable interoperability, and maximize affordability.

Sponsors

- **Presented on behalf of NAVAIR as a consortium member**
- **Work funded by Army PEO Aviation and NAVAIR**

Reusable TS Overview

- **Reusable TS project aims to produce a reusable, portable implementation of the core functionality of the TSS**
- **Reusable TS products can be applied in many software projects using the FACE™ architecture**
- **Eliminates need to re-design and re-implement a core part of the FACE™ architecture (for a wide variety of environments)**
- **Reduces possibility of locking a system to a specific integrator that creates and owns the system's TSS software**

TSS in FACE™ 2.1

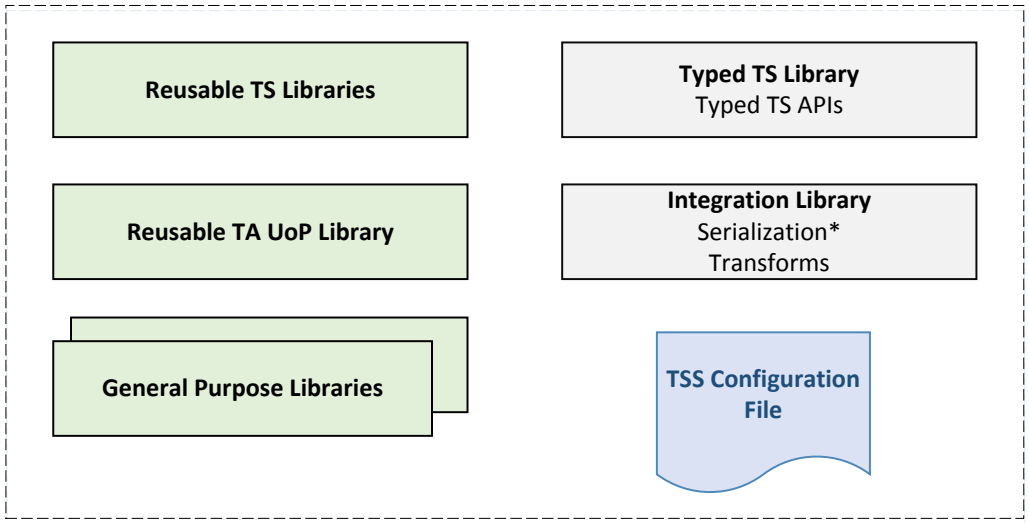
- **Two APIs are standardized:**
  - TS API
  - Type Abstraction (TA) API
- **Other UoCs only see the TS API (PSSS, PCS UoCs)**
- **Three types of Units of Conformance (UoC) – categorized by external interfaces:**
  - TS API that relies on a separate TA API
  - TS API that does not rely on a separate TA API
  - TA API implementation ("TA UoP")

TSS in FACE™ 2.1

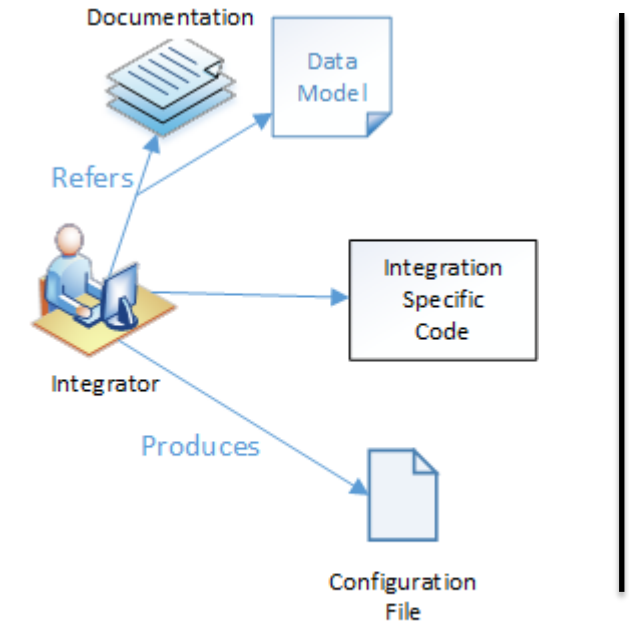
- **TS API include type-specific interfaces**
- **UoC implementation of TS API isn't directly usable between systems with different datatypes**
- **Reusable TS approach:**
  - Implement core capability without referring directly to datatypes
  - Isolate and distribute these capabilities as a package
  - For type specific capabilities, define interfaces internal to the TS that isolate these portions
  - Provide integrator templates and/or generators for type-specific portions

Reusable TS Artifacts

- **Reusable TS products + Type specific code → Complete TS**
- **Configurability provided by configuration file**

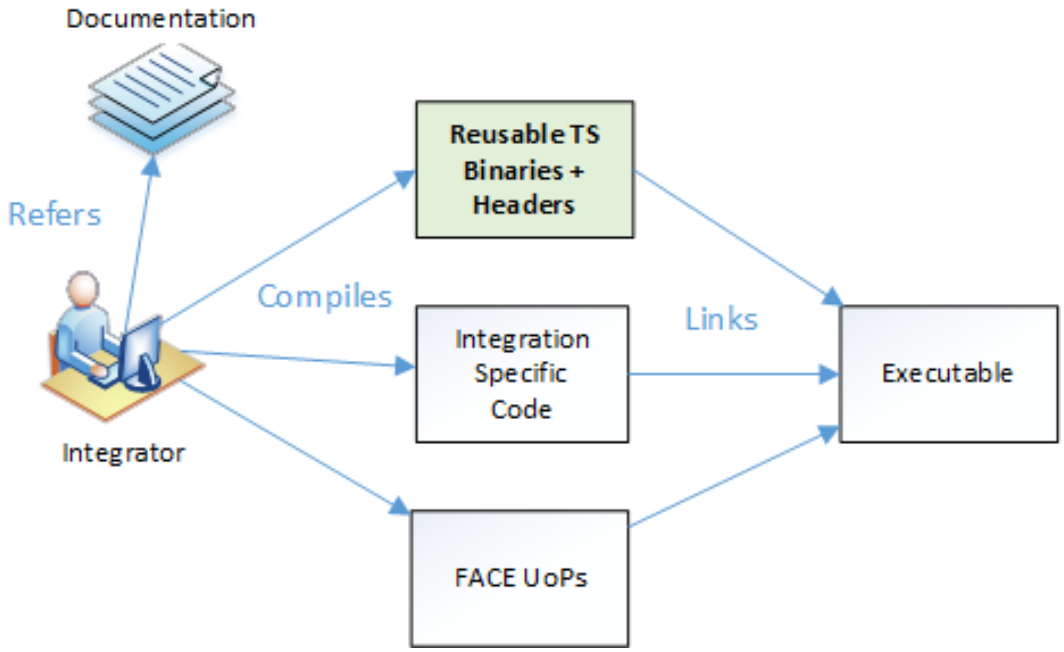


Integrator/Developer Use Cases



*Can be done by generator/script

Integrator Produces type-specific Boilerplate code



Integrator Compiles/Links Code with Reusable TS to produce executable

Distributions of Reusable TS

- **Reusable TS distribution provided as source or binary (for specific environments)**
- **Most code can be distributed as binary and/or does not need to be modified by an integrator**
- **Some code must be provided by integrator – datatype specific boilerplate code (templates and examples provided)**

Configuration Concepts

- Reusable TS configured via configuration file
- Connections – TSS concept; allows UoCs to access TSS
- Subconnections – used to specify individual destinations/sources
- Channels – underlying, system resource

Connection (TS Connection)

- **Identified by a unique name and unique handle identifier (provided by the TS implementation to the application)**
- **Allows application to either send or receive a single datatype**
- **The sources/destinations of that data, and how it is handled are internal to the TS**
- **Application doesn't know the details**
- **Connections map to "message ports" in the FACE™ data model**

Channel (Transport Channel)

- Internal to the Reusable TS
- Abstracts a system resource used for transmitting data
- Is a method of transporting bytes
- Independent of datatype
- Examples are message queues or sockets

Connection / Channel Relationship

- **Integrator maps each connection to one or more channels**
- **Each channel represents a destination or source for data**
- **This mapping is performed in the configuration file**

Channel Configuration

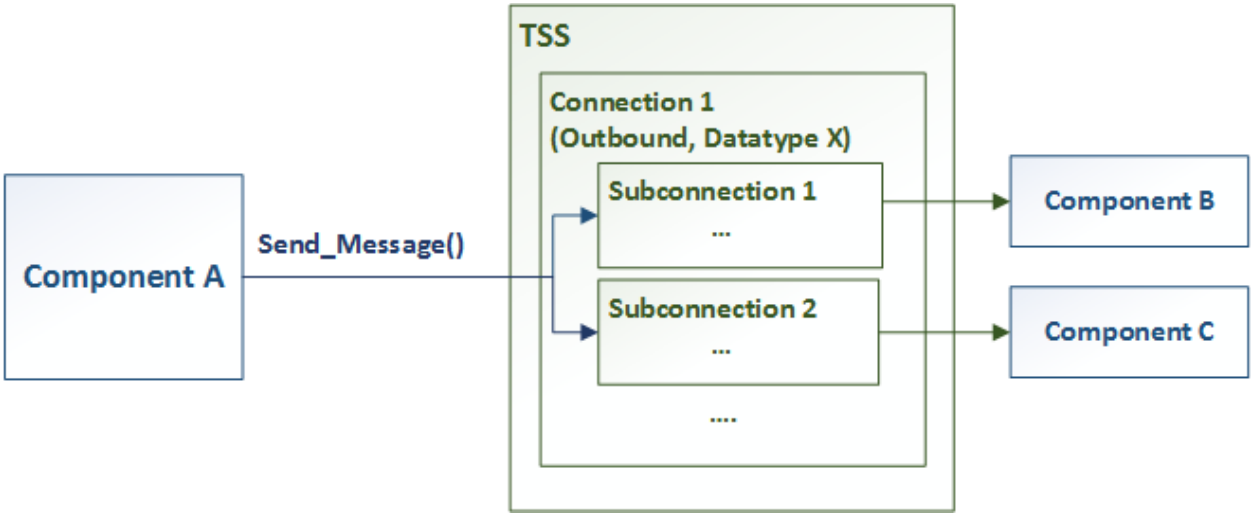
- **Identified by logical name**
- **Types of transport mechanism:**
  - POSIX Message Queues
  - ARINC653 Queuing Ports
  - UDP sockets
  - Data Distribution Service (DDS) (*Using OpenDDS – this method is not currently fully FACE™ aligned)
- **Information specific to the transport**
  - Queue size
  - Queue name
  - IP addresses/ports
  - Etc.

Connection Configuration

- **Connection information is configurable**
  - Name
  - Datatype (identified by unique ID)
  - Component which owns the connection (identified by unique ID)
  - Component on other end of connection (identified by unique ID)
- **Each connection's list of source or destination channel(s) is configurable**
  - Each source/destination is called a **subconnection**
  - Each source or destination is assigned a channel
  - Each source or destination also has an optional transformation sequence and serialization scheme

Channel Configuration Parameters

- **Subconnections determine the distribution of data on a TS connection**
  - One source to one destination
  - One source to many destinations
  - Many sources to one destination



Serialization

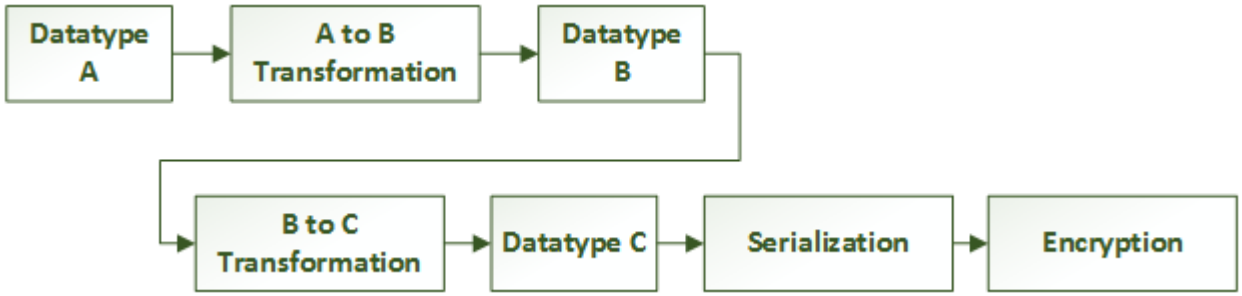
- Serialization is required to send data from one system to another, or to send any data which contains complex types (such as sequences of objects)
- Serialization converts the data into a machine-independent format suitable for transmission
- The receiver uses the reverse process and the known format of the data to deserialize the data back into the datatype
- Serialization is specified for each subconnection

Serialization Design

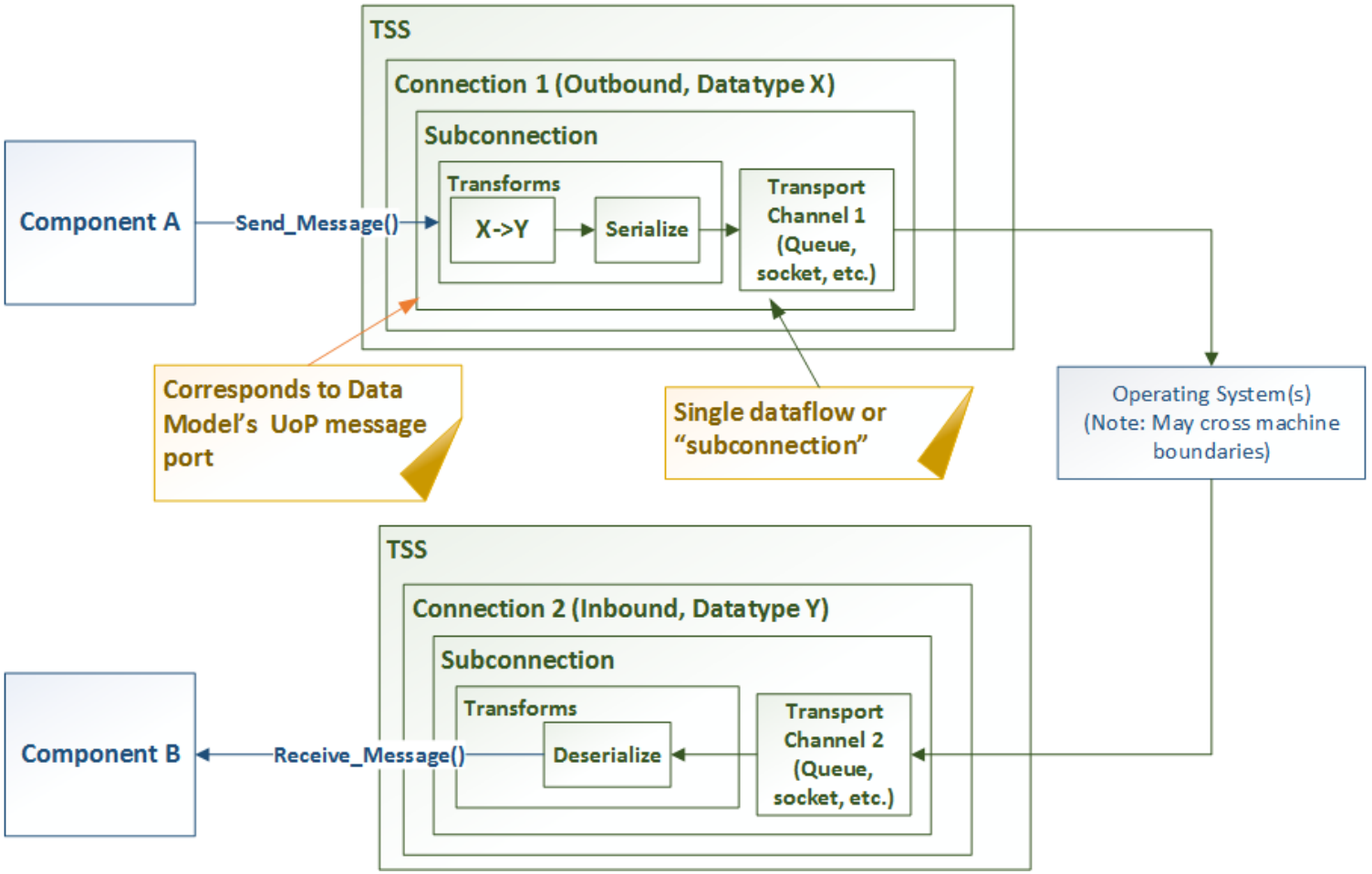
- **Serialization is type specific and therefore generated/ provided by integrator**
- **Reusable TS uses a flexible serialization framework**
  - Supports a packed binary scheme (or no serialization at all – not recommended)
  - Small modifications can allow other styles of serialization
    - XML
    - JSON
    - Key-Value pairs
- **Corinth Tool (provided with Reusable TS) generates serialize and deserialize methods from a FACE™ data model**
  - Methods are separate from actual datatypes

Datatype Transformations

- **Support one-to-one data transformations (Datatype A → Datatype B) on any subconnection**
- **Support multiple transformations**
- **Transformations must be registered in the TSS and linked to the Reusable TSS**
- **A transform sequence is defined in configuration and given a name**
- **Each subconnection can be configured to use a given transform sequences**



Message Pathway



Memory Allocation Scheme (for Safety Base profile only)

- **Reusable TS System Integration and Demo system optionally use a custom memory allocation interface**
- **Required to implement serialization/deserialization of complex types in Safety Base profile**
- **Safety Base profile does not allow deallocation**
- **Replaces type-specific new/delete**
  - FACE Consortium-released Datatype Generator has been updated to produce declarations of new/delete, Corinth tool generates implementations of those functions
- **Global new/delete can also be replaced, but this is experimental and not recommended**

Potential Future Work

- **Update to upcoming edition 3.0 of the FACE™ Technical Standard**
- **Additional transport options and functionality**
- **Integrator tools to generate more of the type specific portions**

Questions