AEROSPACE STANDARD	AS5506/4	VERSION 0.5.0
	Revised September 28, 2019 Superseding Version 0.4.0	

Architecture Analysis & Design Language (AADL)

Annex F: AADL Annex for the FACE^{TM} Technical Standard Edition 3.0

RATIONALE

This annex is intended to provide guidelines for the integrated use of Architecture Analysis & Design Language (AADL) and Future Airborne Capability Environment (FACE) Technical Standard data specifications and components. The FACE Technical Standard Edition 3.0 provides a data modeling specification for software components and their interconnections, but does not, for instance provide mechanisms for describing component behavior or timing properties. This document provides guidance for translating a FACE Standard Edition 3.0 Data Architecture XMI model into AADL so that models of FACE components can be integrated in a standard way into AADL specifications that support AADL analysis and code generation. For example, behavior and timing properties can be added to the resulting model and analyzed using AADL analysis tools.

DISTRIBUTION A. Approved for public release: distribution unlimited.

This material is based upon work supported by U.S. Army Research Development and Engineering Command, Aviation Development Directorate 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 Research Development and Engineering Command, Aviation Development Directorate.

Except for material owned by The Open Group as defined below, Adventium Labs, sole owner of the copyright of this material, hereby grants to the SAE International permission to change, modify, and otherwise utilize materials in this document, in whole or in part, to meet its goals and objectives related to the AADL Standard. Adventium Labs further grants SAE International permission to copyright future versions, including the final standard, as SAE International copyrighted material. This license grant does not extend to, and expressly excludes, materials copyrighted by other parties, such as The Open Group.

Adventium Labs acknowledges The Open Group for permission to include text/figures derived from its copyrighted Future Airborne Capability Environment (FACE) Technical Standard, Edition 3.0, ©2017 The Open Group. FACE™ and the FACE™ logo are trademarks of The Open Group in the United States and other countries.

TO PLACE A DOCUMENT ORDER: **Tel: 877-606-7323 (inside USA and Canada)**
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE WEB ADDRESS: **http://www.sae.org**

SAE values your input. To provide feedback on this Technical Report, please visit http://www.sae.org/technical/standards/PRODCODE

TABLE OF CONTENTS

Copyright 2018 Adventium Labs.

FOREWARD

(1) The Architecture Analysis & Design Language (AADL) standard and its annexes are prepared and updated by the SAE Avionics Systems Division (ASD) Embedded Computing Systems Committee (AS-2) Architecture Description Language (AS-2C) subcommittee.

(2) This AADL standard annex is intended to help component vendors and system integrators using the (Future Airborne Capability Environment) FACE Technical Standard Edition 3.0[1]. FACE Technical Standard Edition 3.0 provides a data modeling architecture but does not provide mechanisms for describing component behavior or timing properties. This document provides guidance for translating a FACE Standard Edition 3.0 Data Architecture XMI model into AADL so that models of FACE components can be integrated in a standard way into AADL specifications that support AADL analysis and code generation[2]. For example, behavior and timing properties can be added to the resulting model and analyzed using AADL analysis tools.

(3) See section J.6 of the FACE Technical Standard Edition 3.0 for Object Constraint Language specifications for the Data Architecture.

[1] Unless explicitly noted, all references to the FACE Technical Standard in this document refer to Edition 3.0.
[2] The FACE Technical Standard Edition 3.0 provides a data architecture metamodel in an EMOF in section J.5.

Copyright 2018 Adventium Labs.

INTRODUCTION

(4) The SAE Architecture Analysis & Design Language (referred to in this document as AADL) is a textual and graphical language used to design and analyze the software and hardware architecture of performance-critical real-time systems. These are systems whose operation strongly depends on meeting non-functional system requirements such as reliability, availability, timing, responsiveness, throughput, safety, and security. AADL is used to describe the structure of such systems as an assembly of software components mapped onto an execution platform. It can be used to describe functional interfaces to components (such as data inputs and outputs) and performance-critical aspects of components (such as timing). AADL can also be used to describe how components interact, such as how data inputs and outputs are connected or how application software components are allocated to execution platform components. The language can also be used to describe the dynamic behavior of the runtime architecture by providing support to model operational modes and mode transitions. The language is designed to be extensible to accommodate analyses of the runtime architectures that the core language does not completely support. Extensions can take the form of new properties and analysis specific notations that can be associated with components and are standardized themselves.

(5) AADL was developed to meet the special needs of performance-critical real-time systems, including embedded real-time systems such as avionics, automotive electronics, or robotics systems. The language can describe important performance-critical aspects such as timing requirements, fault and error behaviors, time and space partitioning, and safety and certification properties. Such a description allows a system designer to perform analyses of the composed components and systems such as system schedulability, sizing analysis, and safety analysis. From these analyses, the designer can evaluate architectural tradeoffs and changes.

(6) AADL supports analysis of cross cutting impact of change in the architecture along multiple analysis dimensions in a consistent manner. Consistency is achieved through automatic generation of analysis models from the annotated architecture model. AADL is designed to be used with generation tools that support the automatic generation of the source code needed to integrate the system components and build a system executive from validated models. This architecture-centric approach to model-based engineering permits incremental validation and verification of system models against requirements and implementations against systems models throughout the development lifecycle.

(7) This document contains the AADL Annex for the FACE Technical Standard Edition 3.0, which guides users in writing or generating AADL models that describe components developed in accordance with the FACE Technical Standard.

Copyright 2018 Adventium Labs.

INFORMATION AND FEEDBACK

(8) The website at http://www.aadl.info is an information source regarding the SAE AADL standard. It makes available papers on AADL, its benefits, and its use. Also available are papers on MetaH, the technology that demonstrated the practicality of a model-based system engineering approach based on architecture description languages for embedded real-time systems.

(9) The website provides links to three SAE AADL related discussion forums:

- The SAE AADL User Forum to ask questions and share experiences about modeling with SAE AADL,
- The AADL Toolset User Forum to ask questions and share experiences with the Open Source AADL Tool Environment, (OSATE) and
- The SAE Standard Document Corrections & Improvements Forum that records errata, corrections, and improvements to the current release of the SAE AADL standard.

(10) The website provides information and a download site for the Open Source AADL Tool Environment. It also provides links to other resources regarding the AADL standard and its use.

(11) Questions and inquiries regarding working versions of annexes and future versions of the standard can be addressed to info@aadl.info.

(12) Informal comments on this standard may be sent via e-mail to errata@aadl.info. If appropriate, the defect correction procedure will be initiated. Comments should use the following format:

!topic Title summarizing comment

!reference AADL-ss.ss(pp)

!from Author Name yy-mm-dd

!keywords keywords related to topic

!discussion

text of discussion

(13) where ss.ss is the section, clause or subclause number, pp is the paragraph or line number where applicable, and yy-mm-dd is the date the comment was sent. The date is optional, as is the !keywords line.

(14) Multiple comments per e-mail message are acceptable. Please use a descriptive “Subject” in your e-mail message.

(15) When correcting typographical errors or making minor wording suggestions, please put the correction directly as the topic of the comment; use square brackets [ ] to indicate text to be omitted and curly braces { } to indicate text to be added, and provide enough context to make the nature of the suggestion self-evident or put additional information in the body of the comment, for example:

!topic [c]{C}haracter

!topic it[']s meaning is not defined

Copyright 2018 Adventium Labs.

Copyright 2018 Adventium Labs.

AADL Annex for the FACE™ Technical Standard, Edition 3.0

Version 0.5.0, 2018-09-27

Typography Conventions

Regular Text
`AADL Keyword`
FACE Keyword Introduction
FACE Keyword

Annex F.1 Scope

(1) This 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.

- This document provides a mapping for FACE Technical Standard Edition 3.0 and AADL 2.2.

(2) This annex includes a FACE property set to be used for common representation of FACE aligned components in AADL models. This document is organized as follows:

- Annex F.2 introduces the FACE Technical Standard and its terminology.
- Annex F.3 introduces the reference model used for examples in this annex.
- Annex F.4 provides recommendations for AADL packaging of models of FACE aligned components.
- Annex F.5 through Annex F.13 provide mappings of FACE Technical Standard elements to AADL.
- Annex F.14 provides recommendations for parsing FACE data model files.
- Annex F.15 provides the AADL property set for models of FACE aligned components.
- Annex F.16 provides guidance on the relationship between the FACE Technical Standard and the AADL runtime services.
- Annex F.17 provides an example of FACE aligned components translated to AADL.

Annex F.2 Background and Assumptions

(1) The FACE Technical Standard provides a framework for data architecture that enables service and application portability across platforms by requiring conformance to the FACE Technical Standard's data modeling and software requirements.

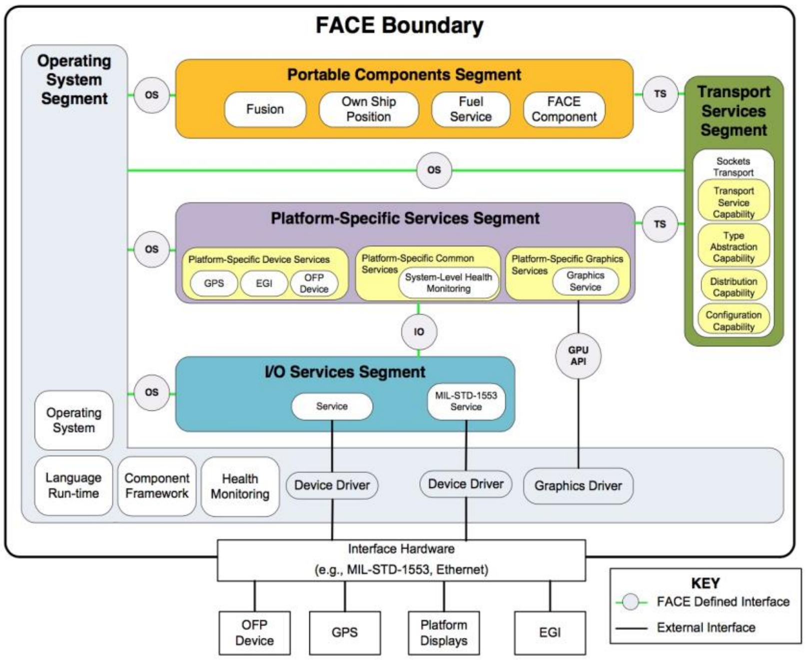

Figure 1 Architecture Segments Example. (Extracted from FACE Technical Standard Edition 3.0 Section 2.4)

Copyright 2018 Adventium Labs.

- As illustrated in Figure 1, the FACE Technical Standard is divided into layers. Individual applications or services that reside in one of these layers are called ***Units of Portability*** (***UoPs***[3]). **UoPs** in the ***Portable Components Segment*** (**PCS**) and the ***Platform Specific Services Segment*** (**PSSS**) communicate with one another using a ***Transport Services*** **Segment** (**TSS**) library. The **PCS** contains general-purpose applications, while the **PSSS** isolates **UoPs** that interact with devices through the **I/O Services Segment (IOSS)**. The **TSS** is an abstract grouping of components (including libraries) that provide data exchange related functionality.
- Communication between **UoPs** is accomplished using parameters dictated by ***views***. **Views** are constructed from a FACE data model using ***queries***.
- In a system built from FACE conformant software, a data architecture is composed by the system integrator using data models associated with each **UoP** in the system.
  i) Systems are not required to consist only of FACE conformant software. The FACE Technical Standard describes conformance criteria for individual components, not for the entire system.
- The fields that make up each **inter-UoP** message are taken from the data model. Each field in each message is associated with a hierarchy of data model elements. This means two **UoPs** that do not need to use precisely the same data representation (e.g., metric or imperial) to communicate with one another, as long as the data representations share a common ancestor.
  i) For further information about the FACE **Data Architecture**, see section 2.3 of the FACE Technical Standard.

(2) The FACE Technical Standard data architecture is divided into three layers: The ***Data Model***, the ***UoP Model***, and the ***Integration Model*** (see Figure 2). This document provides guidance for all three layers.

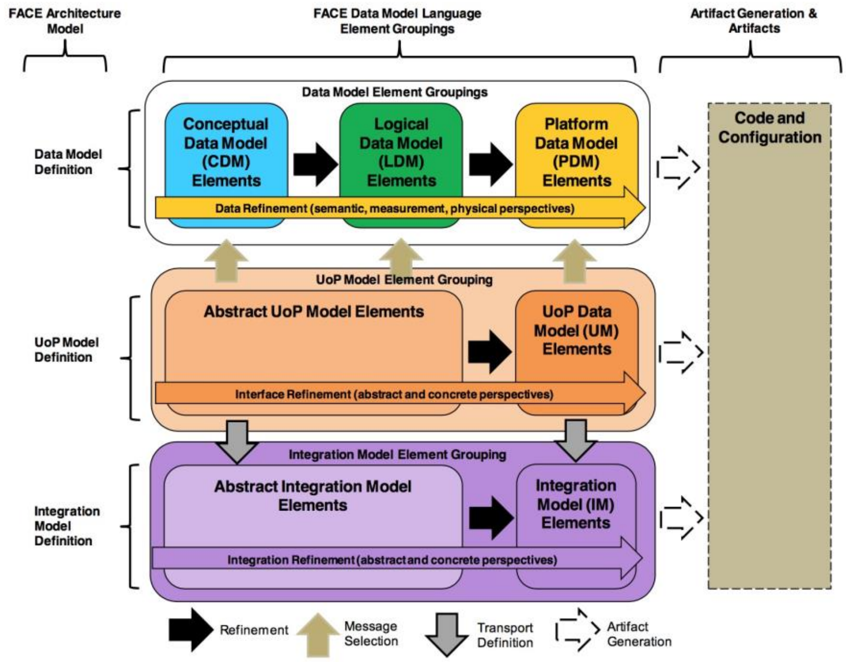

Figure 2 Data Architecture (extracted from FACE Technical Standard Edition 3.0 Section 3.9)

(3) The FACE Technical Standard data model provides a realization hierarchy for multiple levels of data description (*conceptual*, *logical*, and *platform*). Most AADL analyses do not require that multiple levels of the FACE Technical Standard data model are mapped to AADL.

(4) All communication between FACE **UoPs** that reside in the **PCS** or **PSSS** layers is conducted via the **TSS** interface according to **Views** defined in the **Data Model** (as shown in the top and right of Figure 1).

(5) In addition to its data modeling approach to interoperability of **UoPs**, the FACE Technical Standard also provides operating system interface specifications and I/O device interface specifications (see sections 3.2 and 3.4 of the FACE Technical Standard). I/O device access is represented in the FACE **IOSS** (I/O Service Segment). The operating system interface is represented in the FACE **OSS** (Operating System Segment). See the left and bottom of Figure 1.

(6) The terms specific to the FACE Technical Standard used in this annex are defined below:

- **FACE (Future Airborne Capability Environment)**: A government-industry software standard and business strategy for acquisition of affordable software systems that promotes innovation and rapid integration of portable capabilities across global defense programs. The FACE Standard also provides a data modeling language used to describe component interfaces.

[3] The FACE Technical Standard defines two equivalent terms, **Unit of Portability (UoP)** and **Unit of Conformance (UoC)**. This document uses the former, as FACE conformance is not in the scope of this annex.

Copyright 2018 Adventium Labs.

- ***FACE Conformance***: A software component (Unit of Conformance (***UoC***)) is certified as FACE conformant when it has successfully been through an independent verification and certification process, which is defined by the FACE Conformance Program. This includes technical verification by a designated Verification Authority (***VA***) subsequent certification by the FACE Certification Authority (***CA***), and registration in the FACE Library. This certification represents that the software **UoC** meets the requirements of the FACE Technical Standard, which was designed to facilitate software portability. A FACE conformant data architecture is a `.face` file that adheres to the FACE Technical Standard Edition 3.0 metamodel. See section 1.5 of the FACE Technical Standard for more information.
- ***Data Architecture Model*****:** The whole of Figure 2 describes the contents of the data architecture model.
  i) Each system of integrated FACE conformant **UoPs** will ultimately have one **data model**, likely created from multiple input data models.
- ***Data Model***: A set of **conceptual**, **logical**, and **platform** entities used as the basis for **view** definition. Each **platform** entity refines a **logical** entity, and each **logical** entity refines a **conceptual** entity. See top of Figure 2.
  i) **Example**: “Temperature” is <u>conceptual</u>, “Degrees Celsius” is <u>logical</u>, and “32bit unsigned integer” is <u>platform</u>.
- **Domain Specific Data Model (DSDM)**: A **data model** with entities created or refined to address the needs of a particular application or problem space.
- ***UoP Model***: A description of the **UoPs** in a given system of FACE conformant components and their associated views and connections. See middle of Figure 2.
  i) The connections described in the **UoP model** do not describe inter-**UoP** communication. They provide only the **UoP’s** expectations of the type of connection it will have when integrated (e.g., sampling).
  ii) An integrator will combine multiple **UoP models** (one for each integrated **UoP**) into their integrated **UoP model**.
  iii) This term is not equivalent to “USM,” which is defined later in this section.
- ***Integration Model***: A model describing the composition of FACE **UoPs** in a system and the inter-**UoP** message routing in the **TSS**. See bottom of Figure 2.
- ***View***: A FACE **view** is documentation of a **Transport Service** (**TS)** API data parameter that can be passed through the **TSS** via the **TS** interface. A **view** is composed of elements of a **data model** and is described by a **query**.
  i) **Example**: A **view** “status” might include altitude, airspeed, and ground speed.
  ii) **Views** are nominally defined in the **platform** layer of the **Data Model**.
  iii) **Query**: A FACE **query** is an SQL-like expression describing features of the FACE data model to use in a **view**.
  iv) **Template**: A FACE **template** is used to specify the presentation of data in a **platform view**. Table 5 provides an example of a template.
- **UoC (Unit of Conformance)**: A **DSDM** or a software component designed to meet the requirements for an individual FACE segment. **UoCs** must be verified as conformant to the FACE Technical Standard to be certified.
  i) All FACE components in the **PCS**, **TSS**, **PSSS**, and **IOSS** are **UoCs**.
  ii) **UoC** and **UoP** are equivalent terms.
  iii) See section 2.8 of the FACE Technical Standard for more information.
- **UoP (Unit of Portability)**: Also called **Unit of Conformance (UoC)**. Use of the term **Unit of Portability** highlights the portable and reusable attributes of a software component or ***Domain Specific Data Model*** **(DSDM)** developed to the FACE Technical Standard.
  i) Each **UoP** may have an associated **UoP Supplied Model (USM)** providing its **data model** definition and **UoP Model** definition.
- **TSS (Transport Services Segment)**: A **TSS** is responsible for exchanging data between **UoPs**. A **TSS** is also responsible for mediating data between **UoPs** and other data exchange functions.
  i) For example, a **TSS** might translate a “status” parameter to a “heartbeat” parameter with the same fields but different units (perhaps meters instead of feet).
  ii) The **TSS** is often shown as a signal entity in diagrams illustrating systems of FACE conformant software (such as Figure 1) however there is no restriction limiting a system to a single **TSS**.
- ***FACE Shared Data Model***: An instance of a **data model** whose purpose is to define commonly used items and to serve as a basis for all other **data models**.
  i) The FACE **shared data model** provides common concepts such as temperature.
- ***USM (UoP Supplied Model)*****:** A data model provided by a software supplier that documents the data exchanged by a **UoP** via the **TS** interface. An integrated system may incorporate many **USMs**.
  i) The **USM** is provided as a `.face` file with each **UoP**.
- ***Integrated Data Model*****:** The integrator of a system using FACE conformance components combines FACE **USMs** to create an **integrated data model** for the system.
- ***FACE UoP Vendor***: A **UoP** vendor creates the software and **USM** associated with a **UoP.** The **USM** is delivered with the **UoP** software.

Copyright 2018 Adventium Labs.

- ***Integrator of FACE Conformance Components***: The integrator of a system using FACE conformance components is a stakeholder responsible for resolving **USMs** from FACE **UoP** vendors to create the **integrated data model** and for configuring a **TSS** that routes data between **UoPs**.
- ***FACE UUID***: Every element in the **data model** has a unique identifier created using the UUID standard.
  i) UUIDs allow the **Integrator of FACE Conformant Components** to integrate **USMs** from multiple vendors without ambiguity. For example, use of UUIDs mitigates the risks of two **FACE UoP vendors** using the same human-readable name for different components; as each component will have a unique UUID, ambiguity in the human-readable names can be resolved through inspection of the UUIDs.
- ***UoPInstance*****:** A **UoPInstance** is a configuration item describing a **UoP's** role(s) in a given system configuration as described by the **integration model**. A single **UoP** may have multiple instances in a system.
- ***UoPConnection*****:** A **UoPConnection** describes the **UoP's** assumptions about its connection.
  i) A **UoPConnection** does not identify the sender or receiver on the other end of the connection (See Figure 10).
  ii) There are several implementations of UoPConnection, all of which are enumerated in section Annex F.7.
- ***UoPEndPoint*****:** A **UoPEndPoint** describes the routing configuration associated with a single **UoPConnection** (See Figure 10).

Annex F.3 Reference Example

(1) This annex uses the FACE Basic Avionics Lightweight Source Archetype (***BALSA***) example as a point of reference. BALSA source code and FACE models are available to members of The Open Group FACE Consortium.
- Understanding of BALSA is not required to use this annex.

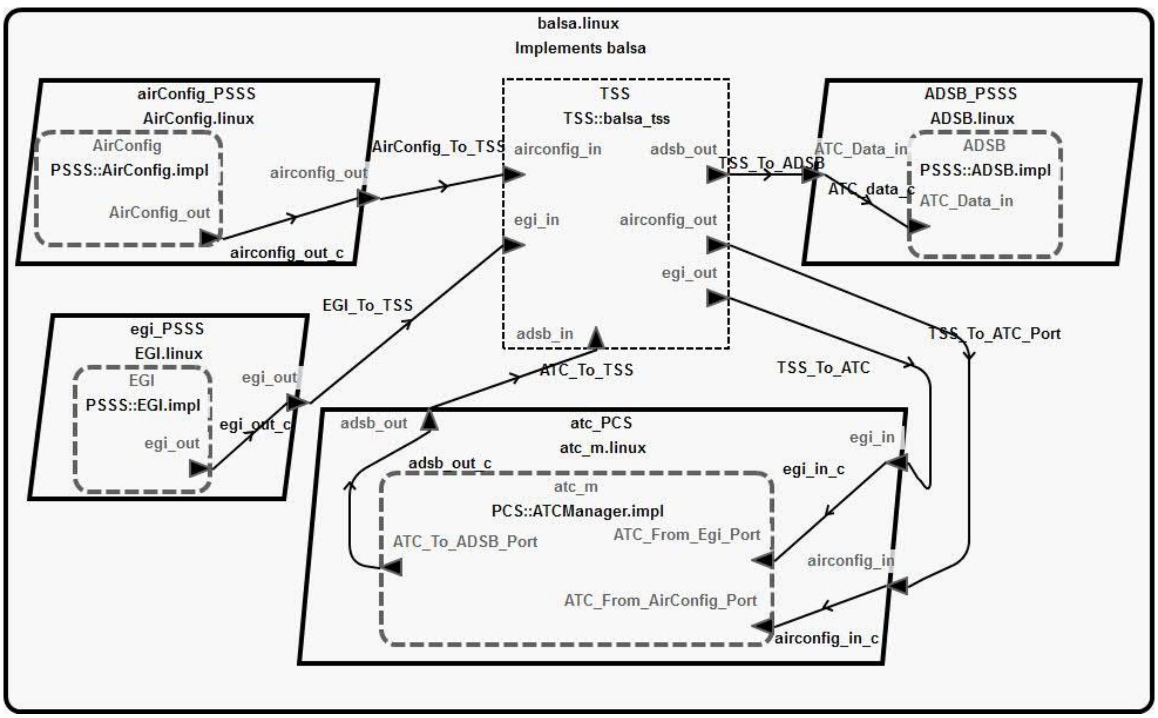

Figure 3: BALSA FACE Integration Model Expressed in AADL

Annex F.4 Packaging

(1) This annex does not provide specific packaging requirements. However, AADL modelers are encouraged to create separate packages for different components of the FACE data model.
- One package for the converted FACE **data model** described with AADL `data` and `data implementations`
- One or more packages for FACE **UoPs** expressed as `thread groups`**.**
- One package for each **integration model**

(2) The **USMs** for each **UoP** will typically contribute both to the data model package and to the UoP package(s).
(3) Example

File	Description	Notes
`data_model.aadl`	`data` and `data implementations` corresponding to FACE **entities** and **views**	

Copyright 2018 Adventium Labs.

`IOSS.aadl`	`thread groups` for **IOSS UoPs**	
`OSS.aadl`	components for the **OSS**	This document does not dictate translation guidelines for the **Operating System Segment**
`PSSS.aadl`	`thread groups` for **PSSS UoPs**	
`PCS.aadl`	`thread groups` for **PCS UoPs**	
`TSS.aadl`	`abstract` defining a **TSS**	
`integration_model.aadl`	`system` and `system implementation` for a system including FACE conformant components. `connections` and `flows` between components.	Optionally includes time and space partitioning via `process` and `virtual processor` in accordance with the integrator's architectural approach

Table 1 Suggested AADL Packaging

Annex F.5 Data Model

(1) The **data model** (top of Figure 2) describes data relevant to a system using FACE conformant components.

- The **system integrator** uses the FACE **Shared Data Model** and **USMs** provided by **UoP** vendors to construct a **data model**.
- **UoP** vendors use or extend the **shared data model**. This means that different **UoPs** will share an ontological heredity between their views, easing the path to translating from one to the other.

(2) Each entity in the **data model** is modeled in AADL as a `data`.

- Modeling the realization hierarchy of **data model** entities is not necessary for most AADL analysis.
- Hierarchical entity representation is modeled using inheritance via the AADL `extends` keyword, as shown in Table 2 and Table 3.

FACE Entity	AADL Entity	Properties
Data Model	`package` (optional)	
Data Model Entity Composition: Conceptual	`data`	• `FACE::UUID` • `FACE::Realization_Tier => conceptual`
Data Model Entity Composition: Logical	`data` or `data extends…`	• `FACE::UUID` • `FACE::Realization_Tier => logical`
Data Model Entity: Platform	`data` or `data extends…`	• `FACE::UUID` • `FACE::Realization_Tier => platform` • `Memory_Properties::Data_Size`

Table 2 FACE Data Model to AADL Mapping

(3) Example

Conceptual	`data aircraftID_Conceptual` `  properties` `    FACE::UUID => "{0540db6f-67fd-430c-bc72-84126daa00cc }";` `    FACE::Realization_Tier => conceptual;` `end aircraftID_Conceptual;`
Logical	`data aircraftID_Logical extends aircraftID_Conceptual` `    properties` `    FACE::UUID => "{ cf4c9604-f2a4-4e38-8937-05fd08e00f0a}";` `    FACE::Realization_Tier => logical;` `end AircraftID_Logical;`
Platform	`data AircraftID_Platform extends aircraftID_logical` `    properties` `    FACE::UUID => "{5e4a3697-13b0-4c35-ba56-29f61f4cdc35}";` `    FACE::Realization_Tier => platform;` `end AircraftID_Platform;`

Table 3 AADL Examples for the FACE Data Model

Copyright 2018 Adventium Labs.

Annex F.6 Data Model Views

(1) A FACE ***Conceptual View*** is either a ***Conceptual Query*** or a ***Conceptual Composite Query***.

- Each **conceptual query** is modeled as a single `data`.
- Each **conceptual composite query** is modeled as a `data` and a `data implementation`.
- The subcomponents of the `data implementation` are determined by the **query compositions** of the **conceptual composite query**.

(2) A FACE ***Logical View*** is either a ***Logical Query*** or a ***Logical Composite Query***.

- Each **logical query** is modeled as a single `data`. If the **logical query** realizes a **conceptual query**, that realization is modeled as a `data extension`.
- Each **Logical Composite Query** is modeled as a `data` and a `data implementation`. If the **logical composite query** realizes a **conceptual composite query**, that realization is modeled as a `data extension`.
- The subcomponents of the `data implementation` are determined by the **Query Compositions** of the **logical composite query**.

(3) A FACE ***Platform View*** is composed of data from the platform tier of the FACE data model.

- A **Platform View** is either a ***Platform Template*** or a ***Platform Composite*** **Template**. A **platform template** has a **bound query**.
- A **Platform View's** contents are defined by a ***query***, the semantics of which are provided in section J.3 of the FACE Technical Standard.
- A **Platform View's** organization is defined by a ***platform template***, the semantics of which are provided in section J.4 of the FACE Technical Standard.
- Each **platform template** is modeled as a single `data`. If the **platform template's bound query** realizes a **logical query**, that realization is modeled as a `data extension`.
- Each **platform composite template** is modeled as a `data` and a `data implementation`. If the **platform composite template** realizes a **logical composite query**, that realization is modeled as a `data extension`.
- The subcomponents of the `data implementation` are determined by the ***Template Compositions*** of the **platform composite template**.
- The `Is_Union` property on an AADL data translated from a FACE Data Model **query** or **template** indicates whether the fields in the **query** or **template** are to be interpreted as a C-style union (if true) or a C-style struct (if false).

FACE Entity	AADL Entity	Properties
Conceptual Query	`data`	• `FACE::UUID` • `FACE::Realization_Tier => Conceptual`
Conceptual Composite Query	`data and data implementation`	• `FACE::UUID` • `FACE::Realization_Tier => Conceptual` • `FACE::Is_Union`
Conceptual Query Composition	`data subcomponent`	• `FACE::UUID`
Logical Query	`data` or `data extends…`	• `FACE::UUID` • `FACE::Realization_Tier => logical`
Logical Composite Query	`data` or `data extends… and data implementation`	• `FACE::UUID` • `FACE::Realization_Tier => logical` • `FACE::Is_Union`
Logical Query Composition	`data subcomponent`	• `FACE::UUID`
Platform Template	`data` or `data extends…`	• `FACE::UUID` • `FACE::Realization_Tier => platform`
Platform Composite Query	`data` or `data extends… and data implementation`	• `FACE::UUID` • `FACE::Realization_Tier => platform`
Platform Composite Template	`data subcomponent`	• `FACE::UUID` • `FACE::Is_Union`

Table 4 Query and Template to AADL Mapping

Copyright 2018 Adventium Labs.

(4) Table 5 shows an example **template** and **query**. The example in Table 6 shows the AADL `data` and `data implementation` for the **composite template** in Table 5.

- Note that in Table 6 the `FACE::UUID` property on `Template_view_from_Aircraft_Config_Platform` refers to the UUID of the template and the `FACE::UUID` properties of the subcomponents of `Template_view_from_Aircraft_Config_Platform` refer to the individual features of the Aircraft **entity**.

Aircraft Entity	<element xsi:type="platform:Entity" xmi:id="_hwTh4EM1EeiBlKadCQCZ8Q" name="Aircraft" description="Aircraft Enitity" realizes="_hwTazkM1EeiBlKadCQCZ8Q"> <composition xmi:id="_hwTh4UM1EeiBlKadCQCZ8Q" rolename="tailNumber" description="tailNumber" type="_hwTh1kM1EeiBlKadCQCZ8Q" realizes="_hwTaz0M1EeiBlKadCQCZ8Q" precision="1000.0"/> <composition xmi:id="_hwTh4kM1EeiBlKadCQCZ8Q" rolename="aircraftID" description="aircraftID" type="_hwTh10M1EeiBlKadCQCZ8Q" realizes="_hwTa0EM1EeiBlKadCQCZ8Q" precision="1000.0"/> </element>
Aircraft Config Template	<element xsi:type="platform:Template" xmi:id="_hwTh8EM1EeiBlKadCQCZ8Q" name="Template_view_from_Aircraft_Config" specification="main (a) {a.aircraftID;a.tailNumber;} " boundQuery="_hwTh70M1EeiBlKadCQCZ8Q" />
Aircraft Config Query	<element xsi:type="platform:Query" xmi:id="_hwTh70M1EeiBlKadCQCZ8Q" name="Aircraft_Config" description="View for message from Aircraft_Config to TSS port" specification="select a.aircraftID, a.tailNumber
from Aircraft as a" />

Table 5 Example Platform Entity, Template, and Query

Platform View	data Template_view_from_Aircraft_Config_Platform properties FACE::Realization_Tier => platform; FACE::UUID => "_hwTh8EM1EeiBlKadCQCZ8Q"; end Template_view_from_Aircraft_Config_Platform; data implementation Template_view_from_Aircraft_Config_Platform.impl subcomponents aircraftID: data AircraftID_Platform { FACE::UUID => "{hwTh4kM1EeiBlKadCQCZ8Q}"; }; tailNumber: data Tail_Number_Platform { FACE::UUID => "{hwTh4UM1EeiBlKadCQCZ8Q}"; }; end Template_view_from_Aircraft_Config_Platform.impl;

Table 6 Example Platform View in AADL

Copyright 2018 Adventium Labs.

Annex F.7 UoP Model

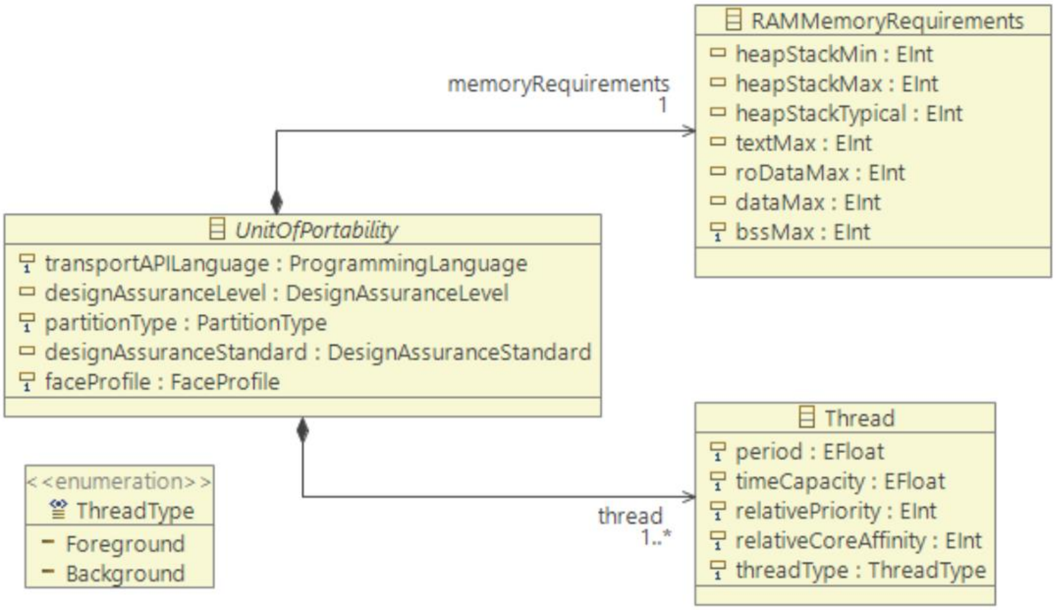

Figure 4 FACE UoP Metamodel Extracted from the FACE Technical Standard Edition 3.0 Section J.2.6

(5) The scope of the FACE **data architecture** is restricted to the data exchanged by software. FACE Technical Standard 3.0 does not describe the physical attributes of a system (e.g., binding software to hardware).

(6) All AADL components translated from FACE **UoPs** use the `FACE::UUID` property to denote the UUID of the FACE component from which they were derived.

- Use of this UUID enables traceability back to the original FACE **USM** from which the AADL component was generated.

(7) A collection of **UoP instances** is modeled as a `system implementation`.

(8) The **UoP model** does not include routing of connections between **UoPs**. Connection routing is described in the **FACE integration model**.

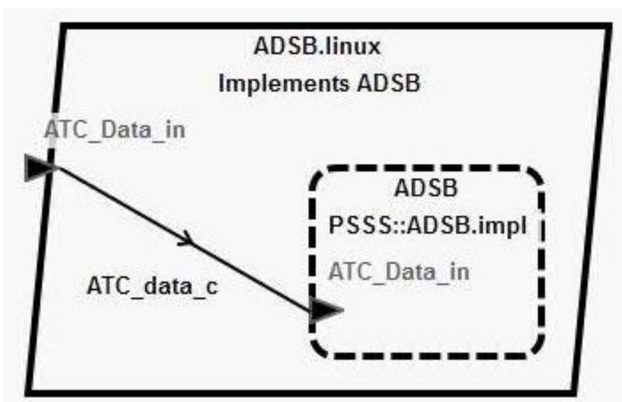

Figure 5: Example UoP (ADSB.impl) Shown Inside a Process (ADSB.linux)

(9) Each FACE **UoP** is modeled in AADL as a `thread group`.

- The FACE Technical Standard does not place requirements on threading of **UoPs**, however the standard does provide for multiple **UoPs** in a single ARINC653 partition or POSIX process. In Figure 5 a single **UoP** is shown inside a process. However, a single `process` could support multiple UoPs. [4]
- Each **UoP** is modeled as a `thread group`.

[4] This annex translates FACE elements to AADL components that can be used in conjunction with a processor and/or virtual processor, thereby permitting but not requiring adherence to ARINC653 or POSIX AADL modeling norms.

Copyright 2018 Adventium Labs.

- A single-threaded **UoP** is modeled as a `thread group` containing a single `thread`. In Figure 5 the **UoP** is called `ADSB`. It is of type `ADSB.impl` and is from the `PSSS` package.
- A multi-threaded **UoP** is modeled as a `thread group` containing multiple `threads`. The FACE Technical Standard metamodel for UoPs is shown in Figure 4.
- **UoPConnections** on the **UoP** are modeled as `ports` on the UoP `thread group`. In Figure 5 the **UoPConnection** is called `ATC_Data_In`.
- AADL `ports` on **UoPs** reference **views** via type constraints.
  i) For example, a **UoPConnection** sending or receiving **Template_view_from_Aircraft_Config** messages (as defined by the FACE **template** in Table 5) would use an AADL port of type `Template_view_from_Aircraft_Config` as defined by the AADL `data` in Table 6.
- The FACE Technical Standard provides several refinements of UoPConnection as shown in Figure 6. The following are the available concrete (non-abstract) connection types:
  i) A ***ClientServerConnection*** is modeled as an `in event data port` and an `out event data port`.[5]
  ii) A ***QueuingConnection*** is modeled as an `in event data port` or an `out event data port`.
  iii) ***SingleInstanceMessageConnection*** is modeled as an `in data port` or an `out data port`.

FACE Entity	AADL Entity	Properties	Notes
UoP	`thread group`	• `FACE::UUID` • `FACE::Segment => PSSS or PCS` • `FACE::Profile`	Can also be modeled as an `abstract`, but `thread group` is preferred.
UoPInstance	`thread group` as `subcomponent`		When a `thread group` is used as subcomponent of a `process`, it is acting as a **UoPInstance**.
UoPConnection	See concrete implementations	• `FACE::UUID` • `Communication_Properties::Input_Rate` and `Communication_Properties::Output_Rate`	The rate of a UoPConnection is specified as a period in seconds in the FACE UoP Model, requiring inversion for representation in AADL.
ClientServerConnection (extends **UoPConnection**)	An `in event data port` with `data type` from associated **view** and an `out event data port` with `data type` from associated **view**		Associated views (**requestType** and **responseType**) are associated with `ports` depending on the **ClientServerRole** property of the connection. If the connection's role is **Client**, then the **requestType view** is associated with the `out port` and the **responseType view** is associated with the `in port`. The association is reversed for **ClientServerConnections** with role **Server**.
QueuingConnection (extends **UoPConnection**)	`in` or `out event data port` with `data type` from associated **view**. The direction of the port is determined by the **MessageExchangeType** property. **InboundMessage** corresponds to an `in port`,	`Communication_Properties::Queue_Size` set from the **queuing connection's depth**	

[5] Although similar to the client server paradigm in intent, AADL `subprogram calls` are not appropriate representations of client server connections as `subprogram calls` imply a remote method invocation paradigm that is not universally consistent with the client server paradigm.

Copyright 2018 Adventium Labs.

	OutboundMessage corresponds to an `out port`.		
SingleInstanceMessageConnection (extends **UoPConnection**)	`in` or `out data port` with `data type` from associated **view**. The direction of the port is determined by the **MessageExchangeType** property. **InboundMessage** corresponds to an `in port`, **OutboundMessage** corresponds to an `out port`.		

Table 7 UoP to AADL Mapping

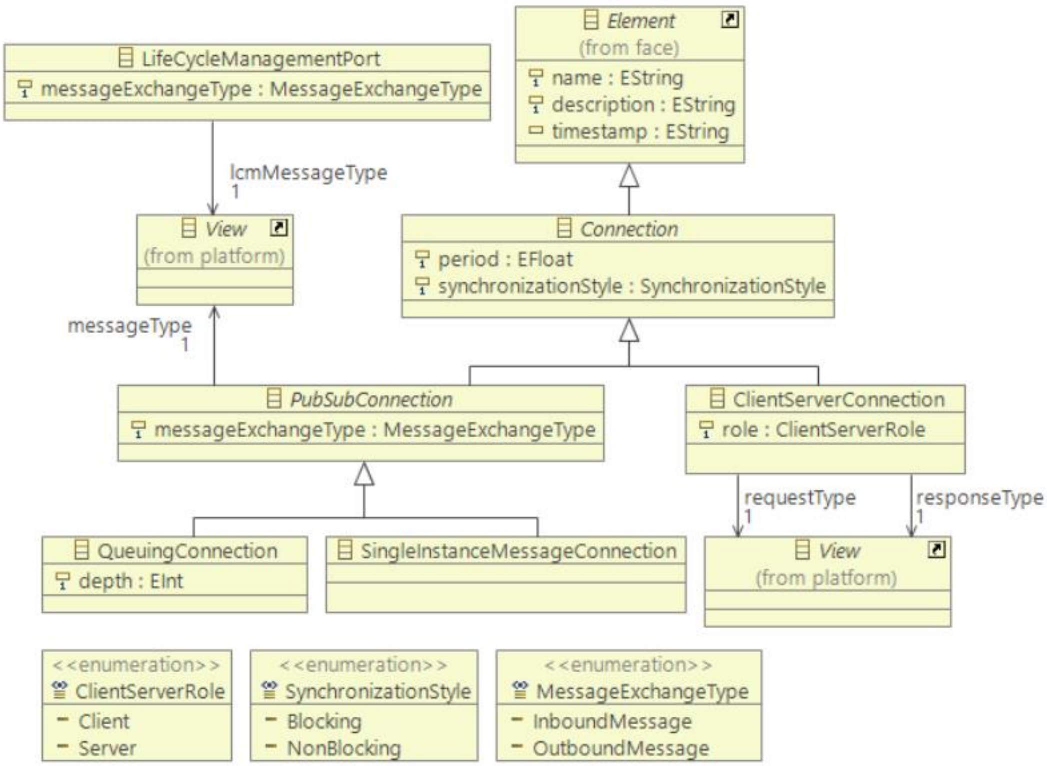

Figure 6 FACE UoP Connections, extracted from the FACE Technical Standard Edition 3.0 Section J.2.6

(10) Each **thread** defined in the **UoP** is modeled as an AADL `thread`.

- The **period** property of the **thread** is assumed to be in seconds and is represented AADL using the `Period` property.
- The **relativePriority** property of the **thread** is assumed to imply higher numerical value means higher priority and is translated directly to the AADL `Priority` property.
- The **timeCapacity** property of the **thread** is assumed to be in seconds and is represented in AADL using the `Compute_Execution_Time` property.
  - i) `Compute_Execution_Time` is a ranged property. Both the minimum and maximum values of `Compute_Execution_Time` should be set to the value of **timeCapacity**.
- `thread` properties not specified by this annex are left to the AADL modeler.

(11) The example shown in Table 8 shows a thread group corresponding to an AirConfig UoP.

Copyright 2018 Adventium Labs.

AirConfig UoP in a FACE Data Model	`<element xsi:type="uop:PlatformSpecificComponent" xmi:id="_hwTh_0M1EeiBlKadCQCZ8Q" name="AirConfig" description="Unit of Portability in PSSS." faceProfile="SafetyBase">` `<thread xmi:id="_hwTiAEM1EeiBlKadCQCZ8Q" period="1.0" timeCapacity="0.1" relativePriority="3"/>` `<memoryRequirements xmi:id="_hwTiAUM1EeiBlKadCQCZ8Q" heapStackMin="1000" heapStackMax="100000" dataMax="100000" bssMax="100"/>` `<connection xsi:type="uop:SingleInstanceMessageConnection" xmi:id="_hwTiAkM1EeiBlKadCQCZ8Q" name="AirConfig_to_ATC_port" description="Interface sending Aircraft_Config data to ATC" period="10.0" synchronizationStyle="NonBlocking" messageType="_hwTh8EM1EeiBlKadCQCZ8Q" messageExchangeType="OutboundMessage"/>`  `</element>`
AirConfig UoP in AADL	`--Unit of Portability in PSSS.` `thread group AirConfig` `features` `--Interface sending Aircraft_Config data to ATC` `AirConfig_to_ATC_port: out data port balsa_data_model::Template_view_from_Aircraft_Config_Platform {` `Output_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit => PerSecond;];` `FACE::UUID => "_hwTiAkM1EeiBlKadCQCZ8Q";` `};` `properties` `FACE::Profile => safety;` `FACE::Segment => PSSS;` `FACE::UUID => "_hwTh_0M1EeiBlKadCQCZ8Q";` `end AirConfig;` `thread group implementation AirConfig.impl` `subcomponents` `thread0: thread {` `Compute_Execution_Time => 1 sec .. 1 sec;` `Period => 1 sec;` `Priority => 3;` `};` `end AirConfig.impl;`

Table 8 Example UoP in AADL

Annex F.8 TSS

(12) A **TSS** is modeled in AADL as an `abstract` that can be `refined` to accommodate varying levels of model detail.

Copyright 2018 Adventium Labs.

- The implementation details for a **TSS** library are not strongly specified by the FACE Technical Standard and may vary from system to system.
    - i) For example, the FACE Technical Standard dictates the semantics of a **Send_Message TSS** function call but does not specify which thread(s) service the call or whether execution of the function requires network access. Such information is available to the system implementers, but the FACE Technical Standard does not provide enough information to specify the appropriate AADL translation a priori.
    - ii) Details about specific TSS implementations should be incorporated into an AADL model as refinements of an `abstract` TSS.
- The FACE Technical Standard's integration model provides *optional* guidelines for configuration of a **TSS**, discussed in section Annex F.9.
- Figure 7 shows an example `system implementation` using only `abstracts` for the **TSS**. Figure 8 shows an extended version of Figure 7 with each `abstract` **TSS** `extended` as a different type (from the top down, `thread group`, `process`, and `system`). The AADL source for these models is in Table 17.

FACE Entity	AADL Entity	Properties
TSS	An `abstract` for each TSS in the system	• `FACE::UUID` • `FACE::Segment=>TSS`
TSS (added detail)	An `abstract` for each TSS in the system implementation, refined as a `virtual bus` (for example)	• `FACE::UUID` • `FACE::Segment=>TSS`
UoP to UoP message route	`flow through` one or more **TSS** `abstract`.	• `FACE::UUID`

Table 9 TSS to AADL Mapping

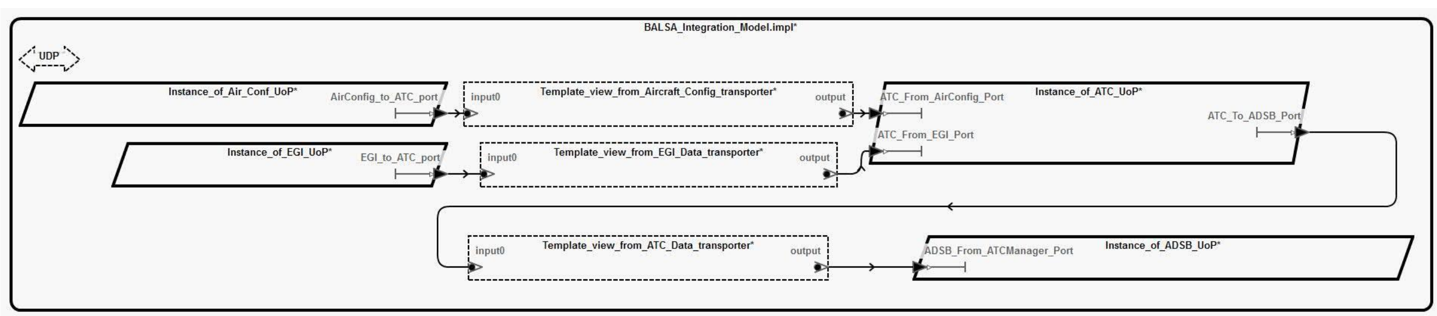

Figure 7 BALSA AADL Model with Abstract TSS Components

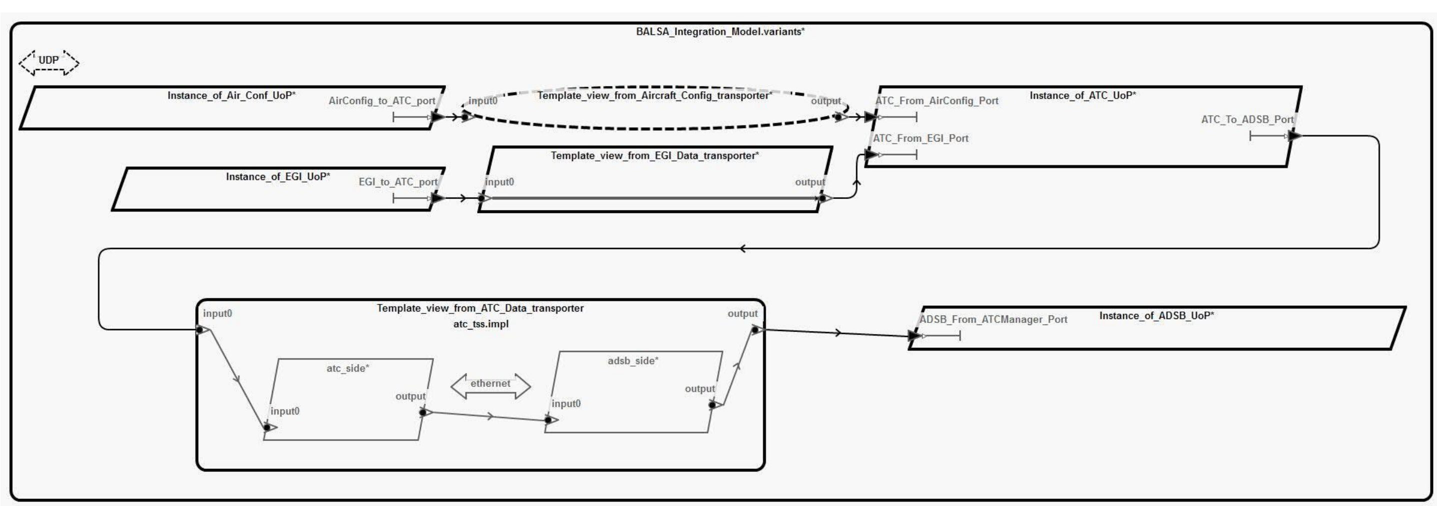

Figure 8 BALSA AADL Model with Different Concrete TSS Components

Copyright 2018 Adventium Labs.

Annex F.9 Routing

(1) The FACE Technical Standard **specifies**, but does not require, a formal model for the configuration of the **TSS** called the ***Integration Model***. The **integration model** includes the routing of data between **UoPs**. Whether or not they opt to use the FACE Technical Standard **integration model**, **system integrators** will have to connect **UoPs**. This annex provides a standard style for their interconnection.

- This document supports use of the FACE **integration model** as specified by the FACE Technical Standard.
- This document provides guidance generally applicable to routing configurations.

(2) The FACE Technical Standard **integration** metamodel provides mechanisms for describing inter-**UoP** communication, including **view** translation (adapting a data interface parameter from one **UoP** to another).

- The entities of the FACE Technical Standard integration metamodel are shown in Figure 9 and Figure 10.

(3) A **UoPInstance** is a **UoP** as used in an **Integration Model**. A single **UoP** may be used multiple times in a FACE **integration model**. The UoP is modeled as a `thread group` and `thread group implementation(s)`. When the **UoP** is used as a `subcomponent`, the `subcomponent` acts as a **UoPInstance**.

- This annex does not specify an AADL representation of the **integration model** as a whole.
- For example, suppose a message logging **UoP** is modeled as a `thread group` named `logger` and implemented as a `thread group implementation` named `logger.impl`. If the FACE Integration Model calls for a **UoPInstance** named my_logger, an AADL `subcomponent` of type `logger.impl` with name `my_logger` should be used. This annex does not specify the parent component of a **UoPInstance**, but an AADL `process` is recommended.

(4) The FACE Technical Standard does not specify organization of **UoPs** into processes. Multiple **UoPs** may be modeled in a single `process` or in multiple `processes`.

(5) A UoP in the **UoP model** defines its **UoPConnections**. These **UoPConnections** are modeled as `ports` in the `thread group` or `thread group implementation`. When the `thread group` is used as a subcomponent, its `ports` act as **UoPEndPoints**.

- A **UoPEndPoint** is a feature of the FACE Technical Standard Integration Model and describes part of the **TSS** configuration. Each **UoPEndPoint** refers to a single **UoPConnection** that it services (see Figure 10).
- Note that a **UoPConnection** is translated to an AADL port, not to an AADL `connection`.
- A **UoPEndPoint** and a **UoPConnection** together define an AADL `port` as used in a **UoPInstance** in a `system implementation`
- AADL `ports` corresponding to **UoPConnections** and **UoPEndPoints** may be organized into `feature groups`.

(6) A **TSNodeConnection** describes the connection from a **UoP** to the **TSS** (not to another UoP)

(7) A **TransportChannel** is modeled as an AADL `virtual bus` to which a **ViewTransporter** is bound. For example, a FACE Integration Model might configure a **view** to be transported between **UoPs** by a **ViewTransporter** and adapted between types using a **ViewTransformation**.

(8) The example in Table 10 shows **UoP** data routing through a **TSS**. `Connections` go from **UoPs** to a **TSS** and `flows` describe data going from **UoP** to **UoP** through the **TSS**.

- The **integration model** alone is insufficient to describe flows that traverse more than two **UoPs**. The end to end flow in Table 10 includes information beyond that provided in the **integration model**; the **integration model** describes the three involved UoPs, each with its connection(s) to the others, but does not describe a data flow through all three.

UoP Routing through TSS	<pre>**connections** AirConfig_To_TSS: **port** airConfig_PSSS.airconfig_out -> TSS.airconfig_in; TSS_To_ATC_Port: **port** TSS.airconfig_out -> atc_PCS.airconfig_in; ATC_To_TSS: **port** atc_PCS.adsb_out -> TSS.adsb_in; TSS_To_ADSB: **port** TSS.adsb_out -> ADSB_PSSS.ATC_Data_in; **flows** AirConfig_ETE: **end to end flow** airconfig_PSSS.AirConfig_Source -> AirConfig_To_TSS -> TSS.AirConfig_flow -> TSS_To_ATC_Port -> atc_PCS.airconfig_adsb_flow -> ATC_To_TSS -> TSS.adsb_flow -> TSS_To_ADSB -> ADSB_PSSS.ATC_Sink;</pre>

Table 10 Example UoP Routing through a TSS

Copyright 2018 Adventium Labs.

FACE Entity	AADL Entity	Properties
Integration Model	`system implementation`	• `FACE::UUID`
UoP Instance	`thread group` as `subcomponent`	• `FACE::UUID`
UoPOutputEndPoint	`port on thread group` as `subcomponent`	• `FACE::UUID`
TSNodePort	`port` on a `TSS abstract`	• `FACE::UUID`
TSNodeConnection	`connection`	• `FACE::UUID`
ViewTransporter	`abstract`	• `FACE::UUID`
TransportChannel	`virtual bus` with **view transporter** `abstract` or **view transporter** `refinement bound` to it	• `FACE::UUID`
ViewFilter, **ViewTransformation**, **ViewAggregation**, **ViewSource**, **ViewSink**	`abstract` to be refined on an implementation-specific basis	• `FACE::UUID`

Table 11 Integration Model to AADL Mapping

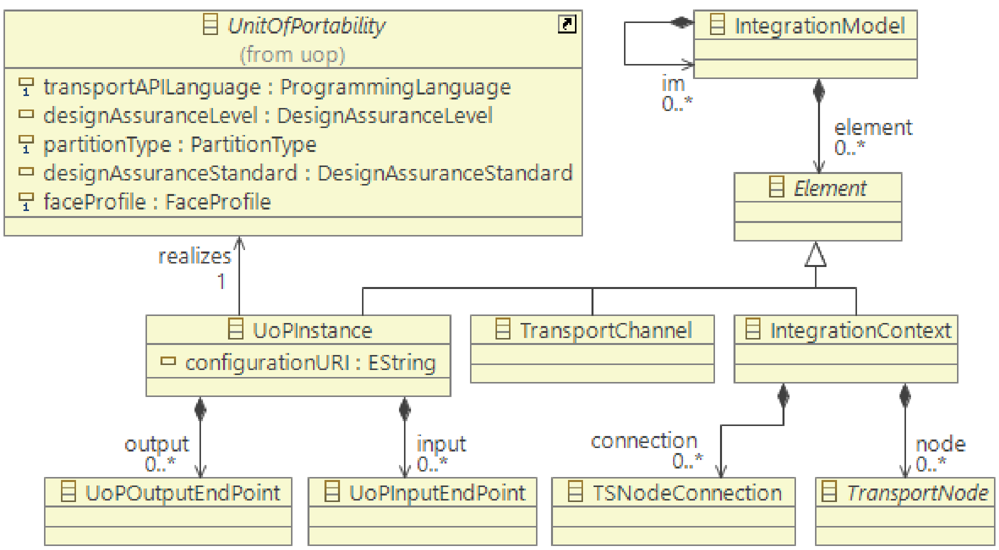

Figure 9 FACE Integration Package, extracted from the FACE Technical Standard Edition 3.0 Section J.2.7

Copyright 2018 Adventium Labs.

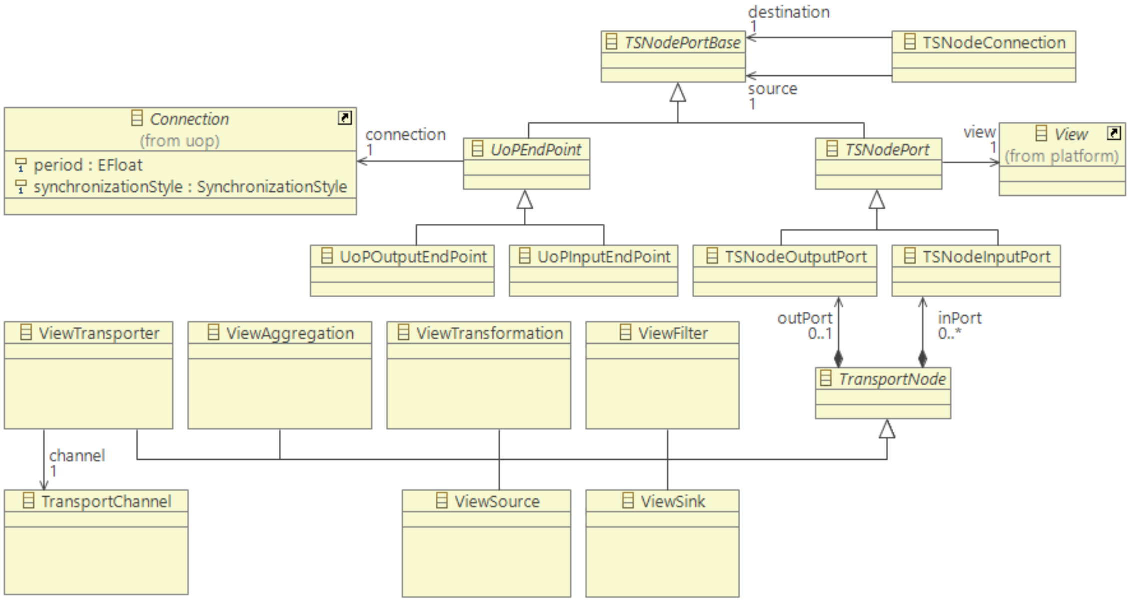

Figure 10 FACE Integration Transport Package, extracted from the FACE Technical Standard Edition 3.0 Section J.2.7

Annex F.10 IOSS

(1) The **IOSS** Layer (bottom of Figure 1) provides an API but does not have a formal exchange model, as **IOSS** components are inherently specific to a particular platform.

- **IOSS** components are modeled in AADL as `abstracts`.
- A **PSSS UoP's** use of **IOSS** functions is modeled in AADL using subprogram calls.
- The physical component to which the **IOSS** service provides access is modeled in AADL as a `device` accessible via a `bus`.
  i) The FACE Technical Standard does not provide means to describe the device itself.
- The bus used by an **IOSS** service to communicate with its physical component(s) is modeled in AADL as a `bus`.
- Table 13 shows an example **IOSS** AADL model. This example opts to extend the **IOSS** `abstract` as a `subprogram` called `ioss1553lib`. Figure 11 shows an AADL graphical model of this example.

FACE Entity	AADL Entity	Properties
IOSS Service	`abstract`	• `FACE::UUID` • `FACE::Profile` • `FACE::Segment=>IOSS`
IOSS Device	`device`	• `FACE::UUID`
IOSS Bus	`bus`	• `FACE::UUID`

Table 12 IOSS to AADL Mapping

Copyright 2018 Adventium Labs.

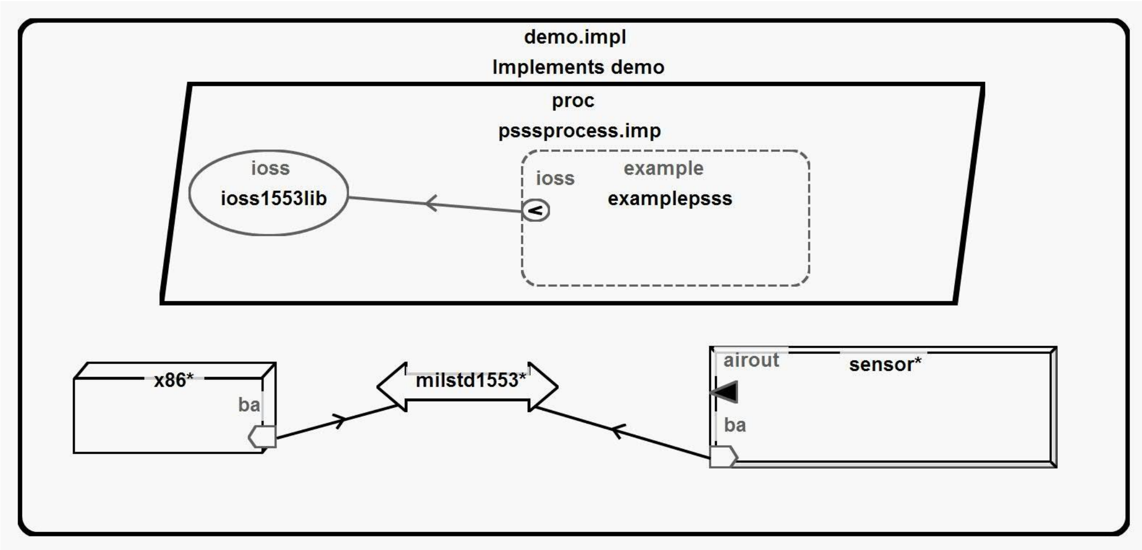

Figure 11 IOSS Example Diagram

<table>
<tr><td>Example IOSS Implementation</td><td>

```
package ioss_example
public

	with FACE;

	bus milstd1553
	end milstd1553;

	abstract ioss1553
		properties
			FACE::Segment => IOSS;
	end ioss1553;

	subprogram ioss1553lib extends ioss1553
	end ioss1553lib;

	thread group examplepsss
		features
			ioss: requires subprogram access ioss1553lib;
		properties
			FACE::Segment => PSSS;
	end examplepsss;

	thread group implementation examplepsss.impl
	end examplepsss.impl;

	process psssprocess
	end psssprocess;

	process implementation psssprocess.impl
		subcomponents
			example: thread group examplepsss;
			ioss: subprogram ioss1553lib;
		connections
			iosscall: subprogram access example.ioss -> ioss;
	end psssprocess.impl;

	device airspeedsensor
		features
			ba: requires bus access milstd1553;
			airout: out data port;
	end airspeedsensor;
```

</td></tr>
</table>

Copyright 2018 Adventium Labs.

	`` system demo end demo; processor x86 features ba: requires bus access milstd1553; end x86; system implementation demo.impl subcomponents x86: processor x86; milstd1553: bus milstd1553; sensor: device airspeedsensor; proc: process psssprocess.impl; connections conn1: bus access x86.ba -> milstd1553; conn2: bus access sensor.ba -> milstd1553; properties Actual_Connection_Binding=>(reference(x86)) applies to proc; end demo.impl; end ioss_example; ```

```
      system demo
      end demo;

      processor x86
            features
                  ba: requires bus access milstd1553;
      end x86;

      system implementation demo.impl
            subcomponents
                  x86: processor x86;
                  milstd1553: bus milstd1553;
                  sensor: device airspeedsensor;
                  proc: process psssprocess.impl;
            connections
                  conn1: bus access x86.ba -> milstd1553;
                  conn2: bus access sensor.ba -> milstd1553;
            properties
                  Actual_Connection_Binding=>(reference(x86)) applies to proc;
      end demo.impl;

end ioss_example;
```

Table 13 Example IOSS AADL Example

Annex F.11 FACE Health Monitoring and Fault Management (HMFM)

(1) The FACE **HMFM** API described in section 3.2.2 of the FACE Technical Standard is out of scope for the current version of this document.

Annex F.12 FACE Profiles

(1) The FACE Technical Standard provides several operating system profiles describing which operating system calls are legal for a **UoC**.

(2) The available profiles (defined in section 2.7 of the FACE Technical Standard) are Security, Safety-Base, Safety-Extended, and General Purpose.

- The `FACE::Profile` property is used to record the profile selected for each component. Appendix A of the FACE Technical Standard enumerates the legal system calls for each profile.
- The FACE::profile restricts the allowed system calls used by generated code to those allowed in a given profile. AADL Tools that generate source code should consider the `FACE::Profile` property when determining how to generate code.

Annex F.13 FACE Lifecycle Management

(1) The FACE **Lifecycle Management** architecture described in section 3.13 of the FACE Technical Standard is out of scope for the current version of this document

- However the Lifecycle Management APIs, States, and Transitions will likely translate naturally to the AADL Behavior Annex and the AADL Runtime Services.
- AADL Tools that generate source code should consider the Lifecycle Management functions described in section 3.13 of the FACE Technical Standard when determining how to generate code.

Annex F.14 FACE Artifact Parsing Guide

(1) The **data model**, **UoP model**, and **integration model** are provided in a standardized Essential Meta-Object Facility (EMOF) format provided in section J.5 of the FACE Technical Standard.

Annex F.15 FACE Property Set

```
-- Properties used when modeling FACE concepts in AADL.
property set FACE is
      -- Properties for all elements:
-- The UUID is used to maintain traceability from elements in the FACE model to elements
in the AADL model.
      UUID: aadlstring applies to (all);
```

Copyright 2018 Adventium Labs.

```
      -- Properties for data model elements:
Realization_Tier: enumeration (conceptual, logical, platform) applies to (data);
-- For data components that correspond to a composite query or a composite template,
Is_Union indicates if the
-- subcomponents should be considered to form a union or a struct.
      Is_Union: aadlboolean => false applies to (data);
      -- Properties for UoPs:
Profile: enumeration (security, safety_extended, safety, general) applies to (all);
Segment: enumeration (PSSS, PCS, IOSS, OSS, TSS) applies to (all);
end FACE;
```

Table 14 AADL Property Set for the FACE Technical Standard Edition 3.0

Annex F.16 Commentary on the AADL Runtime Services

(1) The AADL Runtime Services, defined in section A.9 of the AADL Standard, provide a set of nominal system calls available to threads. Many of these calls overlap in intent with similar functions defined by the FACE Technical Standard. For example `Send_Output` from A.9 (3) of the AADL Standard has similar intent to the **Send_Message** function defined in section E.3.2 of the FACE Technical Standard.

- This annex does not dictate a mapping of AADL Runtime service functions to FACE Technical Standard defined functions.
- This annex does not dictate a mapping of FACE Technical Standard defined functions to AADL Runtime service functions.
- System implementers using both the AADL Runtime Services and a FACE TSS library are encouraged to implement the AADL Runtime Services using a FACE TSS library when applicable. An example of such usage is shown in Figure 12.
    i) Thread behaviors defined using the AADL Behavior Annex can reference the AADL Runtime Services, as the runtime services calls are defined by the AADL Standard. A requirement that AADL behavior specifications reference FACE TSS functions would place an undue burden on tool vendors.

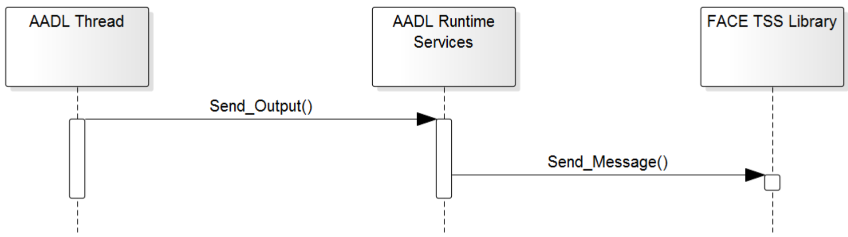

Figure 12 Notional Combined use of AADL Runtime Services and a FACE TSS Library

Annex F.17 BALSA in AADL

BALSA Portable Component Segment UoP in AADL

```
--Generated from "balsa.face" at 2018-09-22T15:52:37.043
package balsa_PCS
public
      with balsa_data_model;
      with FACE;

      --Unit of Portability in PCS
      thread group ATCManager
            features
                  --Interface receiving message from Air_Config
                  ATC_From_AirConfig_Port: in data port
balsa_data_model::Template_view_from_Aircraft_Config_Platform {
                        Input_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh_EM1EeiBlKadCQCZ8Q";
                  };
                  --Interface  receiving message from EGI.
```

Copyright 2018 Adventium Labs.

```
                  ATC_From_EGI_Port: in data port
balsa_data_model::Template_view_from_EGI_Data_Platform {
                        Input_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh_UM1EeiBlKadCQCZ8Q";
                  };
                  --Interface from ATCMAnager sending message to ADSB
                  ATC_To_ADSB_Port: out data port
balsa_data_model::Template_view_from_ATC_Data_Platform {
                        Output_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh_kM1EeiBlKadCQCZ8Q";
                  };
            properties
                  FACE::Profile => safety;
                  FACE::Segment => PCS;
                  FACE::UUID => "_hwTh-UM1EeiBlKadCQCZ8Q";
      end ATCManager;

      thread group implementation ATCManager.impl
            subcomponents
                  thread0: thread {
                        Compute_Execution_Time => 100000001490 ps .. 100000001490 ps;
                        Period => 1 sec;
                        Priority => 3;
                  };
      end ATCManager.impl;

      process ATCManager_process
            features
                  --Interface receiving message from Air_Config
                  ATC_From_AirConfig_Port: in data port
balsa_data_model::Template_view_from_Aircraft_Config_Platform {
                        Input_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh_EM1EeiBlKadCQCZ8Q";
                  };
                  --Interface  receiving message from EGI.
                  ATC_From_EGI_Port: in data port
balsa_data_model::Template_view_from_EGI_Data_Platform {
                        Input_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh_UM1EeiBlKadCQCZ8Q";
                  };
                  --Interface from ATCMAnager sending message to ADSB
                  ATC_To_ADSB_Port: out data port
balsa_data_model::Template_view_from_ATC_Data_Platform {
                        Output_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh_kM1EeiBlKadCQCZ8Q";
                  };
            flows
                  ATC_From_AirConfig_Port_sink: flow sink ATC_From_AirConfig_Port;
                  ATC_From_EGI_Port_sink: flow sink ATC_From_EGI_Port;
                  ATC_To_ADSB_Port_source: flow source ATC_To_ADSB_Port;
      end ATCManager_process;

      process implementation ATCManager_process.impl
            subcomponents
                  ATCManager: thread group ATCManager.impl;
            connections
                  ATC_From_AirConfig_Port_connection: port ATC_From_AirConfig_Port ->
ATCManager.ATC_From_AirConfig_Port;
```

Copyright 2018 Adventium Labs.

```
                  ATC_From_EGI_Port_connection: port ATC_From_EGI_Port ->
ATCManager.ATC_From_EGI_Port;
                  ATC_To_ADSB_Port_connection: port ATCManager.ATC_To_ADSB_Port ->
ATC_To_ADSB_Port;
            flows
                  ATC_From_AirConfig_Port_sink: flow sink ATC_From_AirConfig_Port ->
ATC_From_AirConfig_Port_connection -> ATCManager;
                  ATC_From_EGI_Port_sink: flow sink ATC_From_EGI_Port ->
ATC_From_EGI_Port_connection -> ATCManager;
                  ATC_To_ADSB_Port_source: flow source ATCManager ->
ATC_To_ADSB_Port_connection -> ATC_To_ADSB_Port;
      end ATCManager_process.impl;
end balsa_PCS;
```

Table 15 Example BALSA PCS UoPs Modeled in AADL

BALSA Platform Specific Services Segment in AADL

```
--Generated from "balsa.face" at 2018-09-22T15:52:37.043
package balsa_PSSS
public
      with balsa_data_model;
      with FACE;

      --Unit of Portability in PSSS.
      thread group ADSB
            features
                  --Interface receiving message from ATCManager
                  ADSB_From_ATCManager_Port: in data port
balsa_data_model::Template_view_from_ATC_Data_Platform {
                        Input_Rate => [Value_Range => 0.0 .. 0.0; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh90M1EeiBlKadCQCZ8Q";
                  };
            properties
                  FACE::Profile => safety;
                  FACE::Segment => PSSS;
                  FACE::UUID => "_hwTh9EM1EeiBlKadCQCZ8Q";
      end ADSB;

      thread group implementation ADSB.impl
            subcomponents
                  thread0: thread {
                        Compute_Execution_Time => 100000001490 ps .. 100000001490 ps;
                        Period => 1 sec;
                        Priority => 3;
                  };
      end ADSB.impl;

      process ADSB_process
            features
                  --Interface receiving message from ATCManager
                  ADSB_From_ATCManager_Port: in data port
balsa_data_model::Template_view_from_ATC_Data_Platform {
                        Input_Rate => [Value_Range => 0.0 .. 0.0; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTh90M1EeiBlKadCQCZ8Q";
                  };
            flows
                  ADSB_From_ATCManager_Port_sink: flow sink ADSB_From_ATCManager_Port;
      end ADSB_process;

      process implementation ADSB_process.impl
            subcomponents
```

Copyright 2018 Adventium Labs.

```
			ADSB: thread group ADSB.impl;
		connections
			ADSB_From_ATCManager_Port_connection: port ADSB_From_ATCManager_Port -
> ADSB.ADSB_From_ATCManager_Port;
		flows
			ADSB_From_ATCManager_Port_sink: flow sink ADSB_From_ATCManager_Port ->
ADSB_From_ATCManager_Port_connection -> ADSB;
	end ADSB_process.impl;

	--Unit of Portability in PSSS.
	thread group AirConfig
		features
			--Interface sending Aircraft_Config data to ATC
			AirConfig_to_ATC_port: out data port
balsa_data_model::Template_view_from_Aircraft_Config_Platform {
				Output_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
				FACE::UUID => "_hwTiAkM1EeiBlKadCQCZ8Q";
			};
		properties
			FACE::Profile => safety;
			FACE::Segment => PSSS;
			FACE::UUID => "_hwTh_0M1EeiBlKadCQCZ8Q";
	end AirConfig;

	thread group implementation AirConfig.impl
		subcomponents
			thread0: thread {
				Compute_Execution_Time => 100000001490 ps .. 100000001490 ps;
				Period => 1 sec;
				Priority => 3;
			};
	end AirConfig.impl;

	process AirConfig_process
		features
			--Interface sending Aircraft_Config data to ATC
			AirConfig_to_ATC_port: out data port
balsa_data_model::Template_view_from_Aircraft_Config_Platform {
				Output_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
				FACE::UUID => "_hwTiAkM1EeiBlKadCQCZ8Q";
			};
		flows
			AirConfig_to_ATC_port_source: flow source AirConfig_to_ATC_port;
	end AirConfig_process;

	process implementation AirConfig_process.impl
		subcomponents
			AirConfig: thread group AirConfig.impl;
		connections
			AirConfig_to_ATC_port_connection: port AirConfig.AirConfig_to_ATC_port
-> AirConfig_to_ATC_port;
		flows
			AirConfig_to_ATC_port_source: flow source AirConfig ->
AirConfig_to_ATC_port_connection -> AirConfig_to_ATC_port;
	end AirConfig_process.impl;

	--Unit of Portability in PSSS.
	thread group EGI
		features
			--Interface sending message from EGI to ATCManager
			EGI_to_ATC_port: out data port
```

Copyright 2018 Adventium Labs.

```
balsa_data_model::Template_view_from_EGI_Data_Platform {
                        Output_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTiBkM1EeiBlKadCQCZ8Q";
                  };
            properties
                  FACE::Profile => safety;
                  FACE::Segment => PSSS;
                  FACE::UUID => "_hwTiA0M1EeiBlKadCQCZ8Q";
      end EGI;

      thread group implementation EGI.impl
            subcomponents
                  thread0: thread {
                        Compute_Execution_Time => 100000001490 ps .. 100000001490 ps;
                        Period => 1 sec;
                        Priority => 3;
                  };
      end EGI.impl;

      process EGI_process
            features
                  --Interface sending message from EGI to ATCManager
                  EGI_to_ATC_port: out data port
balsa_data_model::Template_view_from_EGI_Data_Platform {
                        Output_Rate => [Value_Range => 0.1 .. 0.1; Rate_Unit =>
PerSecond;];
                        FACE::UUID => "_hwTiBkM1EeiBlKadCQCZ8Q";
                  };
            flows
                  EGI_to_ATC_port_source: flow source EGI_to_ATC_port;
      end EGI_process;

      process implementation EGI_process.impl
            subcomponents
                  EGI: thread group EGI.impl;
            connections
                  EGI_to_ATC_port_connection: port EGI.EGI_to_ATC_port ->
EGI_to_ATC_port;
            flows
                  EGI_to_ATC_port_source: flow source EGI -> EGI_to_ATC_port_connection
-> EGI_to_ATC_port;
      end EGI_process.impl;
end balsa_PSSS;
```

Table 16 Example BALSA PSSS UoPs Modeled in AADL

BALSA Integration Model Examples in AADL

```
package balsa_integration_model

-- This example package demonstrates approaches
-- for describing a FACE Technical Standard TSS Library
-- in AADL
public
      with balsa_data_model;
      with balsa_PCS;
      with balsa_PSSS;
      with FACE;

      -- The AADL Annex for the FACE Technical Standard Edition 3.0
      -- Dictates that TSS libraries are modeled as abstracts and
      -- extended to meet the needs of the modeler(s).
      abstract Template_view_from_EGI_Data_transporter
```

Copyright 2018 Adventium Labs.

```
            features
                  input0: in feature
balsa_data_model::Template_view_from_EGI_Data_Platform {
                        FACE::UUID => "_hwdLZEM1EeiBlKadCQCZ8Q";
                  };
                  output: out feature
balsa_data_model::Template_view_from_EGI_Data_Platform {
                        FACE::UUID => "_hwdLY0M1EeiBlKadCQCZ8Q";
                  };
            properties
                  FACE::Segment => TSS;
                  FACE::UUID => "_hwdLYkM1EeiBlKadCQCZ8Q";
      end Template_view_from_EGI_Data_transporter;


      abstract Template_view_from_ATC_Data_transporter
            features
                  input0: in feature
balsa_data_model::Template_view_from_ATC_Data_Platform {
                        FACE::UUID => "_hwdLZ0M1EeiBlKadCQCZ8Q";
                  };
                  output: out feature
balsa_data_model::Template_view_from_ATC_Data_Platform {
                        FACE::UUID => "_hwdLZkM1EeiBlKadCQCZ8Q";
                  };
            properties
                  FACE::Segment => TSS;
                  FACE::UUID => "_hwdLZUM1EeiBlKadCQCZ8Q";
      end Template_view_from_ATC_Data_transporter;

      abstract Template_view_from_Aircraft_Config_transporter
            features
                  input0: in feature
balsa_data_model::Template_view_from_Aircraft_Config_Platform {
                        FACE::UUID => "_hwdLakM1EeiBlKadCQCZ8Q";
                  };
                  output: out feature
balsa_data_model::Template_view_from_Aircraft_Config_Platform {
                        FACE::UUID => "_hwdLaUM1EeiBlKadCQCZ8Q";
                  };
            properties
                  FACE::Segment => TSS;
                  FACE::UUID => "_hwdLaEM1EeiBlKadCQCZ8Q";
      end Template_view_from_Aircraft_Config_transporter;

      system BALSA_Integration_Model
            properties
                  FACE::UUID => "_hwdLUEM1EeiBlKadCQCZ8Q";
      end BALSA_Integration_Model;

      -- This system implementation uses only the abstract representations of TSS
libraries.
      system implementation BALSA_Integration_Model.impl
            subcomponents
                  --A UoP instance is analogous to the use of a thread group as a
subcomponent
                  Instance_of_ATC_UoP: process balsa_PCS::ATCManager_process.impl {
                        FACE::UUID => "_hwdLUUM1EeiBlKadCQCZ8Q";
                  };
                  --A UoP instance is analogous to the use of a thread group as a
subcomponent
                  Instance_of_EGI_UoP: process balsa_PSSS::EGI_process.impl {
                        FACE::UUID => "_hwdLVUM1EeiBlKadCQCZ8Q";
```

Copyright 2018 Adventium Labs.

```
			};
			--A UoP instance is analogous to the use of a thread group as a 
subcomponent
			Instance_of_Air_Conf_UoP: process balsa_PSSS::AirConfig_process.impl {
				FACE::UUID => "_hwdLV0M1EeiBlKadCQCZ8Q";
			};
			--A UoP instance is analogous to the use of a thread group as a 
subcomponent
			Instance_of_ADSB_UoP: process balsa_PSSS::ADSB_process.impl {
				FACE::UUID => "_hwdLWUM1EeiBlKadCQCZ8Q";
			};
			--Use a subclass of transport channel instead for more details
			UDP: virtual bus {
				FACE::UUID => "_hwdLa0M1EeiBlKadCQCZ8Q";
			};
			Template_view_from_EGI_Data_transporter: abstract 
Template_view_from_EGI_Data_transporter;
			Template_view_from_ATC_Data_transporter: abstract 
Template_view_from_ATC_Data_transporter;
			Template_view_from_Aircraft_Config_transporter: abstract 
Template_view_from_Aircraft_Config_transporter;
		connections
			connection0: feature Instance_of_EGI_UoP.EGI_to_ATC_port -> 
Template_view_from_EGI_Data_transporter.input0 {
				FACE::UUID => "_hwdLXEM1EeiBlKadCQCZ8Q";
			};
			connection1: feature Template_view_from_EGI_Data_transporter.output -> 
Instance_of_ATC_UoP.ATC_From_EGI_Port {
				FACE::UUID => "_hwdLXUM1EeiBlKadCQCZ8Q";
			};
			connection2: feature Instance_of_ATC_UoP.ATC_To_ADSB_Port -> 
Template_view_from_ATC_Data_transporter.input0 {
				FACE::UUID => "_hwdLXkM1EeiBlKadCQCZ8Q";
			};
			connection3: feature Template_view_from_ATC_Data_transporter.output -> 
Instance_of_ADSB_UoP.ADSB_From_ATCManager_Port {
				FACE::UUID => "_hwdLX0M1EeiBlKadCQCZ8Q";
			};
			connection4: feature Instance_of_Air_Conf_UoP.AirConfig_to_ATC_port -> 
Template_view_from_Aircraft_Config_transporter.input0 {
				FACE::UUID => "_hwdLYEM1EeiBlKadCQCZ8Q";
			};
			connection5: feature 
Template_view_from_Aircraft_Config_transporter.output -> 
Instance_of_ATC_UoP.ATC_From_AirConfig_Port {
				FACE::UUID => "_hwdLYUM1EeiBlKadCQCZ8Q";
			};
	end BALSA_Integration_Model.impl;

	-- The abstract TSS can be extended as a separate process and thread
	-- You might use this paradigm if send and receive message calls
	-- are serviced by a different thread from the calling UoP
	-- and you wish to analyze latency from one UoP to another through the TSS 
library.
	thread Template_view_from_EGI_Data_transporter_thread extends 
Template_view_from_EGI_Data_transporter
		flows
			through: flow path input0 -> output;
	end Template_view_from_EGI_Data_transporter_thread;

	thread implementation Template_view_from_EGI_Data_transporter_thread.impl
	end Template_view_from_EGI_Data_transporter_thread.impl;
```

Copyright 2018 Adventium Labs.

```
	-- An AADL process provides a memory space
	process egi_tss extends Template_view_from_EGI_Data_transporter
		flows
			through: flow path input0 -> output;
	end egi_tss;

	process implementation egi_tss.impl
		subcomponents
			t1: thread Template_view_from_EGI_Data_transporter_thread;
	end egi_tss.impl;

	-- The abstract TSS can alternatively be as subprogram group
	-- Subprograms in AADL are analogous to functions in most programming languages
	-- A subprogram group is analogous to a library of functions.
	-- You might use this paradigm if you want to generate source code for UoPs
	-- using AADL code generation tools.
	-- Note that this example shows only send_message and receive_message, omitting 
several
	-- TSS functions and type specifiers on non-message parameters for brevity.
	subprogram send_message
		-- Parameters per E.3.2 of the FACE Technical Standard
		features
			connection_id: in parameter;
			timeout: in parameter;
			transaction_id: in out parameter;
			message: in parameter 
balsa_data_model::Template_view_from_Aircraft_Config_Platform;
			return_code: out parameter;
	end send_message;

	subprogram receive_message
		-- Parameters per E.3.2 of the FACE Technical Standard
		features
			connection_id: in parameter;
			timeout: in parameter;
			transaction_id: in out parameter;
			message: in out parameter 
balsa_data_model::Template_view_from_Aircraft_Config_Platform;
			header: out parameter;
			qos_parameters: out parameter;
			return_code: out parameter;
	end receive_message;

	subprogram group airconfig_tss extends 
Template_view_from_Aircraft_Config_transporter
		features
			send_message: provides subprogram access send_message;
			receive_message: provides subprogram access receive_message;
	end airconfig_tss;

	subprogram group implementation airconfig_tss.impl
	end airconfig_tss.impl;

	-- The abstract TSS can be extended as separate system.
	-- This example shows a TSS library that uses a physical bus
	-- to transport data.
	system atc_tss extends Template_view_from_ATC_Data_transporter
	end atc_tss;

	-- The TSS system has two processes, one for each side of the communication
	process atc_side_tss extends Template_view_from_ATC_Data_transporter
	end atc_side_tss;
```

Copyright 2018 Adventium Labs.

```
      process adsb_side_tss extends Template_view_from_ATC_Data_transporter
      end adsb_side_tss;

      system implementation atc_tss.impl
            -- Use a prototype to delay specification of the bus
            prototypes
                  bus_proto : bus;
            subcomponents
                  atc_side : process atc_side_tss;
                  adsb_side : process adsb_side_tss;
                  b : bus  bus_proto;
            connections
                  conn1 : feature input0 -> atc_side.input0;
                  conn2 : feature adsb_side.output -> output;
                  throughput1: feature atc_side.output -> adsb_side.input0;
            properties
                  Actual_Connection_Binding => (reference(b)) applies to throughput1;
      end atc_tss.impl;

      -- Define a physical bus to be specified for the TSS-as-a-system
      bus ethernet
      end ethernet;

      system implementation BALSA_Integration_Model.variants
            subcomponents
                  --A UoP instance is analogous to the use of a thread group as a
subcomponent
                  Instance_of_ATC_UoP: process balsa_PCS::ATCManager_process.impl {
                        FACE::UUID => "_hwdLUUM1EeiBlKadCQCZ8Q";
                  };
                  --A UoP instance is analogous to the use of a thread group as a
subcomponent
                  Instance_of_EGI_UoP: process balsa_PSSS::EGI_process.impl {
                        FACE::UUID => "_hwdLVUM1EeiBlKadCQCZ8Q";
                  };
                  --A UoP instance is analogous to the use of a thread group as a
subcomponent
                  Instance_of_Air_Conf_UoP: process balsa_PSSS::AirConfig_process.impl {
                        FACE::UUID => "_hwdLV0M1EeiBlKadCQCZ8Q";
                  };
                  --A UoP instance is analogous to the use of a thread group as a
subcomponent
                  Instance_of_ADSB_UoP: process balsa_PSSS::ADSB_process.impl {
                        FACE::UUID => "_hwdLWUM1EeiBlKadCQCZ8Q";
                  };

                  -- TSS implemented as a separate process
                  Template_view_from_EGI_Data_transporter: process egi_tss.impl;

                  -- TSS implemented as subprogram group
                  Template_view_from_Aircraft_Config_transporter: subprogram group
airconfig_tss.impl;

                  -- TSS implemented as system (note specification for the bus
prototype)
                  Template_view_from_ATC_Data_transporter: system atc_tss.impl
(bus_proto => bus ethernet);

                  --Use a subclass of transport channel instead for more details
                  UDP: virtual bus {
                        FACE::UUID => "_hwdLa0M1EeiBlKadCQCZ8Q";
                  };
            connections
```

Copyright 2018 Adventium Labs.

```
                  connection0: feature Instance_of_EGI_UoP.EGI_to_ATC_port ->
Template_view_from_EGI_Data_transporter.input0 {
                        FACE::UUID => "_hwdLXEM1EeiBlKadCQCZ8Q";
                  };
                  connection1: feature Template_view_from_EGI_Data_transporter.output ->
Instance_of_ATC_UoP.ATC_From_EGI_Port {
                        FACE::UUID => "_hwdLXUM1EeiBlKadCQCZ8Q";
                  };
                  connection2: feature Instance_of_ATC_UoP.ATC_To_ADSB_Port ->
Template_view_from_ATC_Data_transporter.input0 {
                        FACE::UUID => "_hwdLXkM1EeiBlKadCQCZ8Q";
                  };
                  connection3: feature Template_view_from_ATC_Data_transporter.output ->
Instance_of_ADSB_UoP.ADSB_From_ATCManager_Port {
                        FACE::UUID => "_hwdLX0M1EeiBlKadCQCZ8Q";
                  };
                  connection4: feature Instance_of_Air_Conf_UoP.AirConfig_to_ATC_port ->
Template_view_from_Aircraft_Config_transporter.input0 {
                        FACE::UUID => "_hwdLYEM1EeiBlKadCQCZ8Q";
                  };
                  connection5: feature
Template_view_from_Aircraft_Config_transporter.output ->
Instance_of_ATC_UoP.ATC_From_AirConfig_Port {
                        FACE::UUID => "_hwdLYUM1EeiBlKadCQCZ8Q";
                  };
      end BALSA_Integration_Model.variants;


end balsa_integration_model;
```

Table 17 Example BALSA TSS Variations Modeled in AADL

Copyright 2018 Adventium Labs.