AADL ANNEX FOR THE FACE™ TECHNICAL STANDARD, EDITION 3.0

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.

AADL Annex for the FACE™ Technical Standard, Edition 3.0

Version 0.3.0, 2018-04-04

Typography Conventions

Regular Text
`AADL Keyword`
FACE Keyword Introduction
FACE Keyword

A. Rationale

1) This annex is intended to help component vendors and system integrators using the (Future Airborne Capability Environment) FACE Technical Standard. FACE Technical Standard Edition 3.0 [1] 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[2] into AADL so that behavior and timing properties can be added and analyzed.
   a) See section J.6 of the FACE Technical Standard Edition 3.0 for Object Constraint Language specifications for the Data Architecture.
2) 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.

B. Background and Assumptions

3) This document provides a mapping for FACE Technical Standard Edition 3.0 and AADL 2.2.
4) The FACE Technical Standard provides a framework for data architecture that enables service and application portability across platforms by requiring

[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.

conformance to the FACE Technical Standard's data modeling and software guidelines.

a) As illustrated in Figure 2, 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 Service* Segment (**TSS**) library. The **PCS** contains general-purpose applications, while the **PSSS** isolates **UoPs** that interact with devices through the **I/O segment (IOS)**. The **TSS** is an abstract grouping of components (including libraries) that provide data exchange related functionality.
b) Communication between **UoPs** is accomplished using parameters dictated by ***views***. **Views** and are constructed from a FACE data model using ***queries***.
c) In a system built from FACE conformant software, there is a single data architecture model. This data architecture model is composed by the system integrator using data models associated with each **UoP** in the system.
d) 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.
    i) For further information about the FACE **Data Architecture**, see section 2.3 of the FACE Technical Standard.

5) The FACE Technical Standard data architecture is divided into three layers: The ***Data Model***, ***UoP Model***, and the ***Integration Model*** (see Figure 1). This document provides guidance for all three.

[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.

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

a)

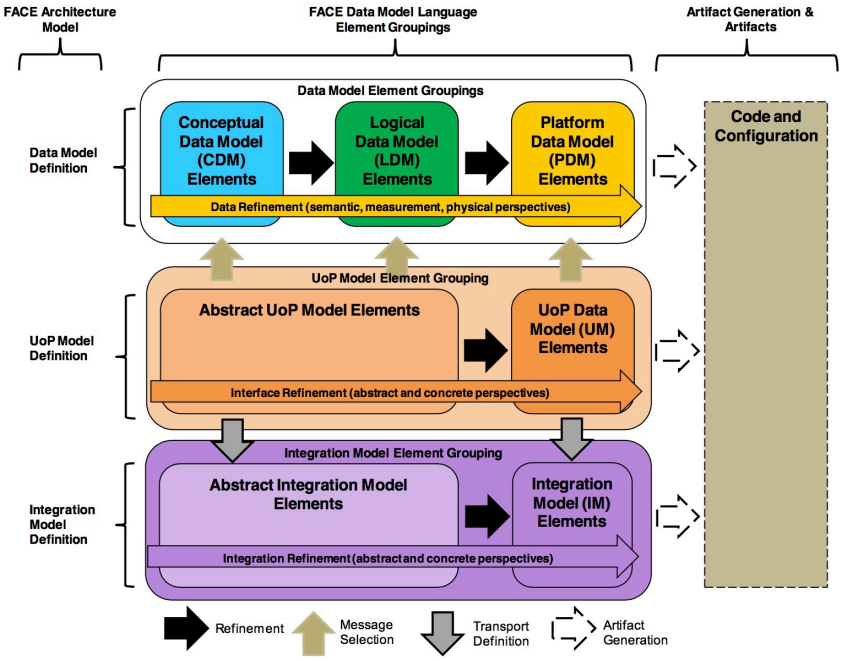
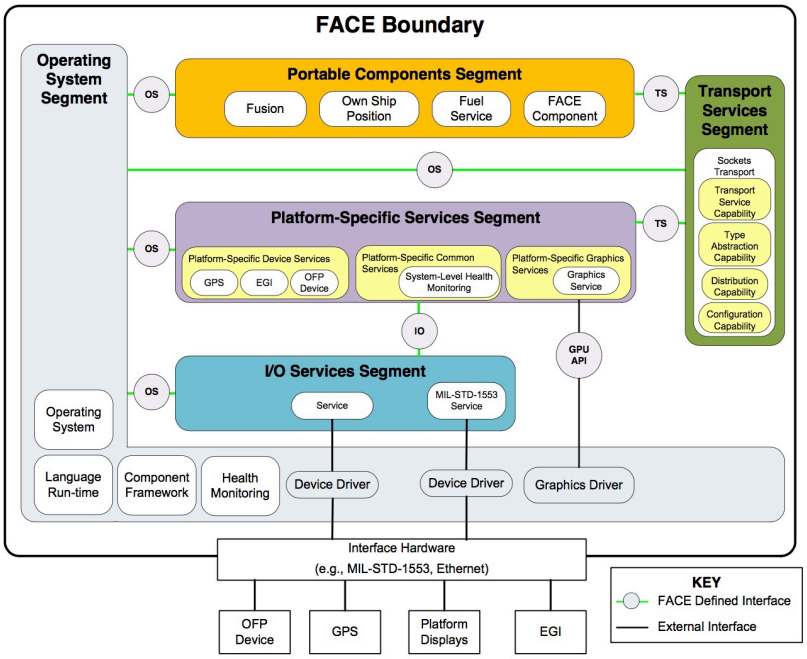


Figure 1 Data Architecture (extracted from FACE Technical Standard Edition 3.0)

7) All communication between the 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 2).
8) 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. 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 2.



a)

Figure 2 Architecture Segments Example. (Extracted from FACE Technical Standard Edition 3.0)

9) The terms specific to the FACE Technical Standard used in this annex are defined below:
   a) **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.
   b) **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.

c) **Data Architecture Model:** The whole of Figure 1 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.

d) **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 1.
   i) **Example**: "Temperature" is conceptual, "Degrees Celsius" is logical, and "32bit unsigned integer" is platform.

e) **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 1.
   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.

f) **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 1.

g) **View**: A FACE **view** is documentation of a **TS** API data parameter that can be passed in 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**.
   v) **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.
   vi) Each **UoP** may have an associated **USM** providing its **data model** definition and **UoP Model** definition.

h) **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.

i) **TSS (Transport Service 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 2) however there is no restriction limiting a system to a single **TSS**.
j) **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.
k) **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**.
l) **Integrated Data Model:** The integrator of a system using FACE conformance components combines FACE **USMs** to create the **Integrated Data Model** for the system.
m) **FACE UoP Vendor**: A **UoP** vendor creates the software, **data model**, and **UoP model** associated with a **UoP**. The **data model** and **UoP model** are delivered with the **UoP** software.
n) **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 and for configuring a **TSS** that routes data between **UoPs**.
o) **FACE UUID**: Every element in the **Data Model** has a unique identifier created using the UUID standard.
p) **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.
q) **UoPConnection:** A **UoPConnection** describes the **UoP's** assumptions about its connection. A **UoPConnection** does not identify the sender or receiver on the other end of the connection (See Figure 7).
r) **UoPEndPoint:** A **UoPEndPoint** describes the routing configuration associated with a single **UoPConnection** (See Figure 7).

C. Reference Example

10) 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.
    a) Understanding of BALSA is not required to use this annex.

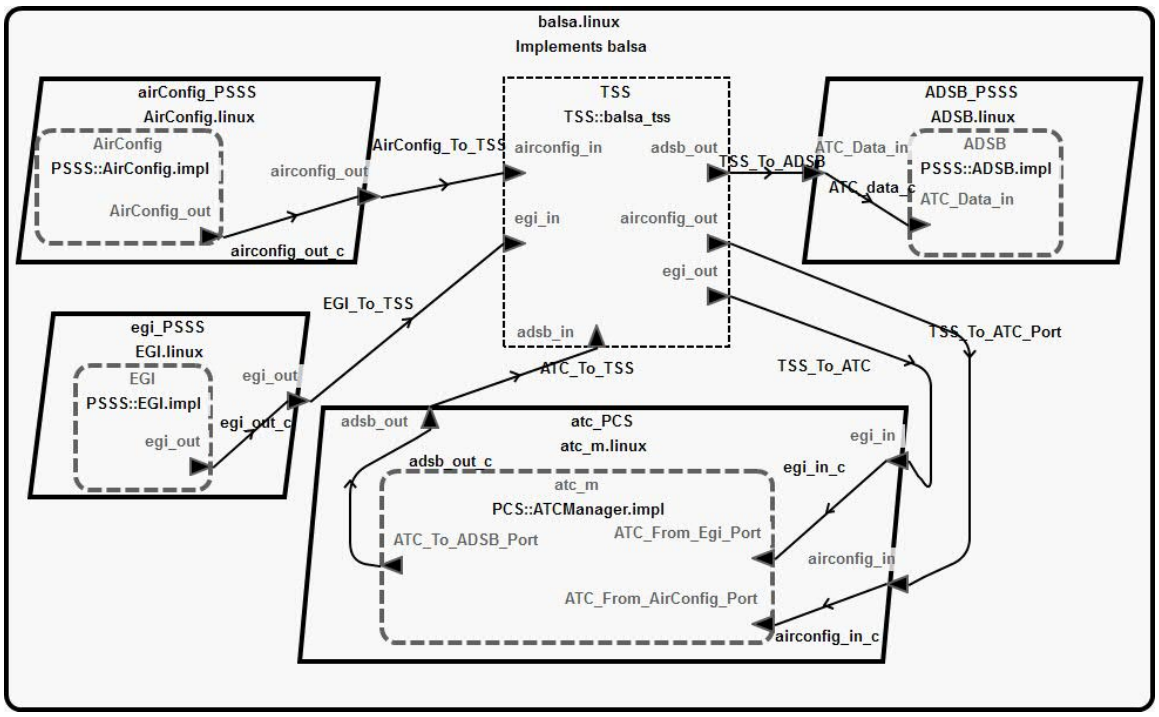


Figure 3: BALSA modeled in AADL

D. Packaging

11) This annex does not provide specific packaging requirements. However, modelers are encouraged to create separate packages.
    a) One package for the **Data Model**
    b) One or more packages for **UoPs**
    c) One package for each **Integration Model**
12) The **USMs** for each **UoP** will contribute *both* to the Data Model package and to the **UoP** package(s).
13) Example

File	Description	Notes
`data_model.aadl`	`data` and `data implementations` corresponding to FACE **entities** and **views**	
`IOS.aadl`	`thread groups` for **IOS UoCs**	
`OSS.aadl`	components for the **OSS**	
`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	Optionally includes time and space partitioning via process and virtual processor

E. Data Model

14) The **Data Model** (top of Figure 1) describes data relevant to a system using FACE conformant components.
   a) The **System Integrator** uses the FACE **Shared Data Model** and **USMs** provided by **UoP** vendors to construct a **Data Model**.
   b) **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.
15) Each entity in the **Data Model** is modeled in AADL as a data.
   a) Modeling the realization hierarchy of **Data Model** entities is not necessary for most AADL analysis.

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

16) 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;`

F. Data Model Views

17) A FACE **Platform View** is composed of data from the platform tier of the FACE data model.
   a) 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.
   b) A **Platform View's** organization is defined by a **template**, the semantics of which are provided in section J.4 of the FACE Technical Standard.
   c) Each **Platform View** is modeled as a single `data implementation`.
   d) The subcomponents of the `data implementation` are determined by the **Platform View's template** and that **template's boundQuery**.

FACE Entity	AADL Entity	Properties
Conceptual View	`data` and `data implementation`	• `FACE::UUID` • `FACE::Realization_Tier => Conceptual`
Logical View	`data` and `data implementation`	• `FACE::UUID` • `FACE::Realization_Tier => logical`
Platform View	`data` and `data implementation`	• `FACE::UUID` • `FACE::Realization_Tier => platform`

18) The example in Table 1 shows the AADL `data` and `data implementation` for a **template** and its **boundQuery** that include an `aircraftID` and `tailNumber`.

Platform View	`data aircraft_config`
	`end aircraft_config;`
	
	`data implementation aircraft_config.impl`
	`  subcomponents`
	`  aircraftID: data AircraftID_Platform;`
	`  tailNumber: data Tail_Number_Platform;`
	`  properties`
	`  FACE::Realization_Tier => platform;`
	`end aircraft_config.impl;`

Table 1 Example platform view in AADL

G. UoP Model

19) The scope of the FACE **Data Architecture** is restricted to the data exchanged by software. FACE 3.0 does not describe the physical attributes of a system (e.g., binding hardware to software).
20) 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.
21) A collection of **UoP Instances** is modeled as a `system implementation`.
22) The **UoP model** does not include routing of connections between **UoPs**. Connection routing is described in the **FACE Integration Model**.

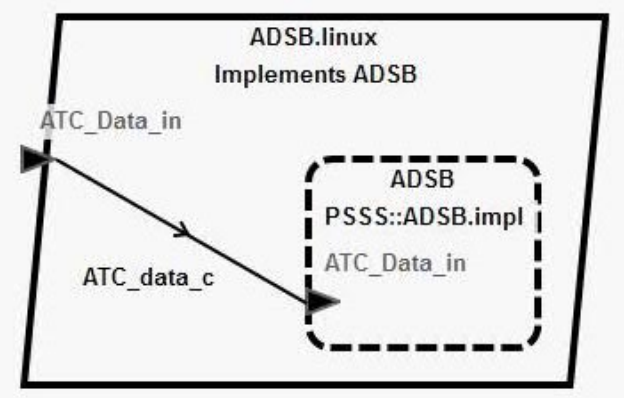


Figure 4: Example UoP (ADSB.impl) shown inside a process (ADSB.linux)

23) Each FACE UoP is modeled in AADL as a `thread group`.
    a) 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 4 a single **UoP** is shown inside a process. However, a single `process` could support multiple UoPs. [4]
    b) A single-threaded **UoP** is modeled as a `thread group` containing a single `thread`. In Figure 4 the **UoP** is called `ADSB`. It is of type `ADSB.impl` and is from the `PSSS` package.
    c) **UoPConnections** on the **UoP** are modeled as `ports` on the UoP `thread group`. In Figure 4 the **UoPConnection** is called `ATC_Data_In`.
    d) AADL `ports` on **UoPs** reference **Views** via type constraints.
    e) The FACE Technical Standard provides several refinements of UoPConnection as shown in Figure 5. 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`.

[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.

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::FaceSegment => 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` • `FACE::ViewUUID` • `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`, **OutboundMessage** corresponds to an `out port`.	`Communication_Properties::Queue_Size` set from **Depth**	
SingleInstanceMessageConnetion (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`, **OutboundMessa**		

	ge corresponds to an `out port`.		

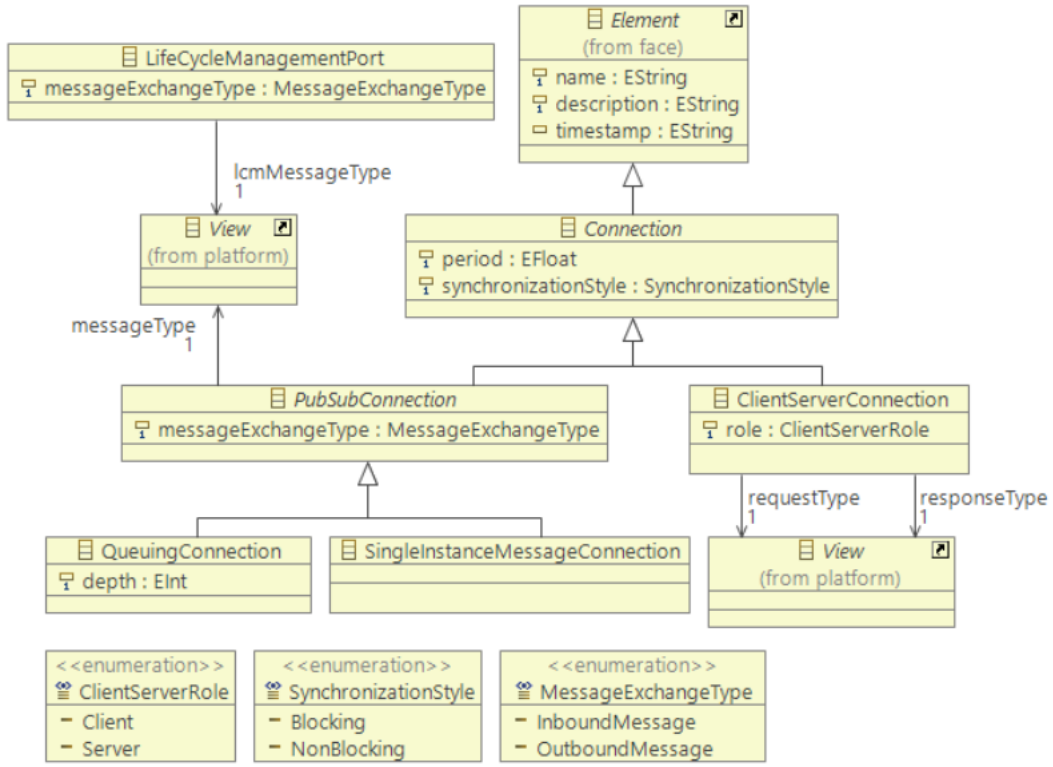


Figure 5 FACE UoP Connections, extracted from the FACE Technical Standard Edition 3.0

24) Each **thread** defined in the **UoP** is modeled as an AADL `thread`.
   a) The **period** property of the **thread** is assumed to be in seconds and is represented AADL using the `Period` property.
   b) 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.
   c) The **timeCapacity** property of the **thread** is assumed to be in seconds and is represented in AADL using the `Compute_Execution_Time` property.
25) The example shown in Table 2 shows a thread group corresponding to an Automatic Dependent Surveillance-Broadcast (ADSB) UoP.

UoP	**thread group** ADSB **features** ADSB_From_ATCManager_Port: **in data port** balsa_data_model::atc_data.impl; **properties**

	FACE::UUID => "{5884a330-a191-498a-9378-11b61f3c1c77}"; FACE::FaceSegment => PCS; **end** ADSB;

Table 2 Example UoP in AADL

H. TSS

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

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`

I. Routing

27) 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.
    a) This document supports use of the FACE **Integration Model** as specified by the FACE Technical Standard.
    b) This document provides guidance generally applicable to routing configurations.
28) 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).

a) The entities of the FACE Technical Standard integration metamodel are shown in Figure 6 and Figure 7.

29) 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**.
    a) This annex does not specify an AADL representation of the **Integration Model** as a whole.
    b) 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.
30) 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`.
31) 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` used as a subcomponent, its `ports` act as **UoPEndPoints**.
    a) 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 7).
    b) Note that a **UoPConnection** is *not* equivalent to an AADL `connection`.
    c) Note that a **UoPEndPoint** is not directly equivalent to an AADL `port`. A **UoPEndPoint** and a **UoPConnection** together define an AADL `port`.
    d) AADL `ports` corresponding to **UoPConnections** may be organized into `feature groups`.
32) A **TSNodeConnection** describes the connection from a **UoP** to the **TSS** (not to another UoP)
33) 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**.
34) The example in Table 3 shows **UoP** data routing through a **TSS**. `Connections` go from **UoPs** to a **TSS** and `flows` describe data going from **UoP** to **UoP**.
    a) The **Integration Model** alone is insufficient to describe flows that traverse more than two **UoPs**. The flows in Table 3 include information beyond that provided in the **Integration Model**.

UoP Routing through TSS	**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;

Table 3 Example UoP Routing through a TSS

FACE Entity	AADL Entity	Properties
Integration Model	system implementation	• FACE::UUID

FACE Entity	AADL Entity	Properties
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	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

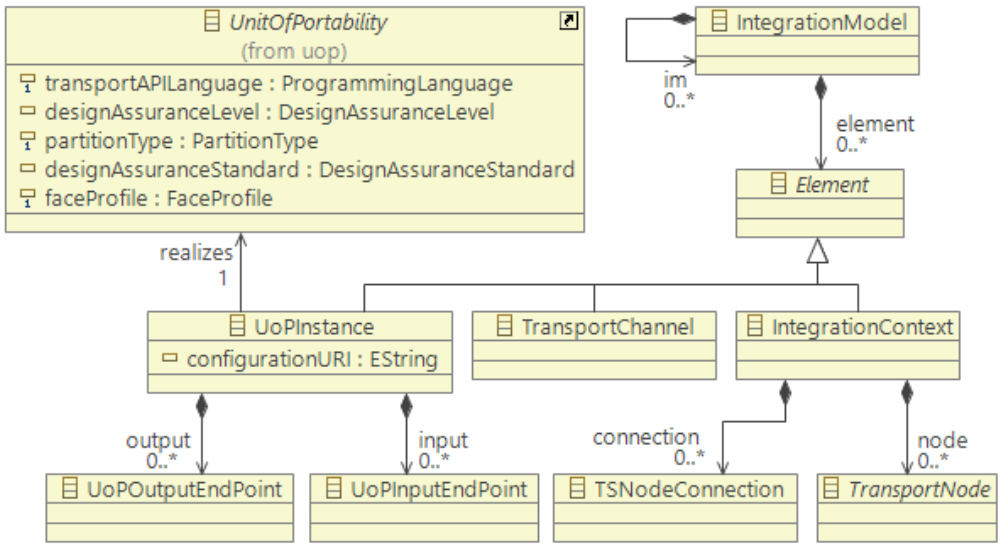


Figure 6 FACE Integration Package, extracted from the FACE Technical Standard Edition 3.0

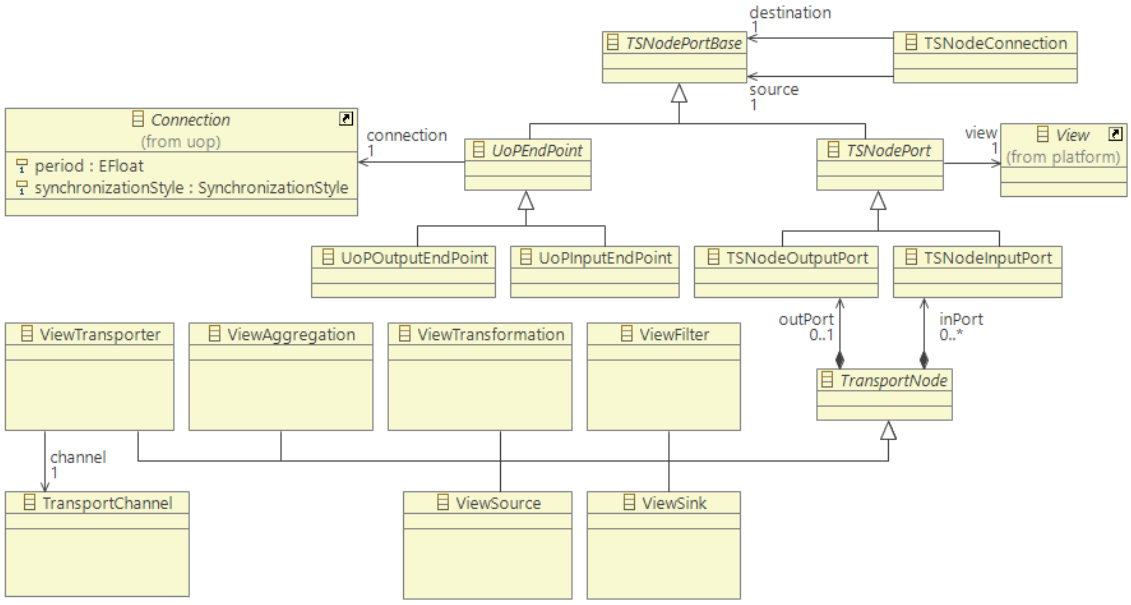


Figure 7 FACE Integration Transport Package, extracted from the FACE Technical Standard Edition 3.0

J. IOSS

35) The **IOSS** Layer (bottom of Figure 2) provides an API but does not have a formal exchange model, as **IOSS** components are inherently specific to a particular platform.

a) **IOSS** components are modeled in AADL as `abstracts`.
b) A **PSSS UoP's** use of **IOSS** functions is modeled in AADL using subprogram calls.
c) The physical component to which the **IOSS** service provides access is modeled in AADL as a `device`.
d) The bus used by the **IOSS** service to communicate with its physical component(s) is modeled in AADL as a `bus access`.

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

K. FACE Health Monitoring and Fault Management (HMFM)

36) The FACE **HMFM** API is a subset of the ARINC653 HMFM API, which is described in the AADL ARINC653 annex.

L. FACE Profiles

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

M. FACE Lifecycle Management

38) The FACE Lifecycle Management architecture 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.

N. FACE Artifact Parsing Guide

39) The **Data Model**, **UoP Model**, and **Integration Model** are provided in a standardized EMOF format provided in section J.5 of the FACE Technical Standard.

O. FACE Property Set

```
property set FACE is
   Profile: enumeration (security, safety_extended, safety, general)
applies to (all);
   Tier: type enumeration (conceptual, logical, platform);
   UUID: aadlstring applies to (all);
   Realization_Tier: FACE::Tier applies to (all);
   segment: type enumeration (PSSS, PCS, IOSS, OSS, TSS);
   FaceSegment: FACE::segment applies to (all);
end FACE;
```