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.

AADL Annex for the FACE™ Technical Standard, Edition 3.0 3

Typography Conventions 3

A. Rationale 3

B. Background and Assumptions 3

C. Reference Example 8

D. Packaging 9

E. Data Model 10

F. Data Model Views 11

G. UoP Model 12

H. TSS 13

I. Routing 14

J. IOSS 16

K. FACE Health Monitoring and Fault Management (HMFM) 17

L. FACE Profiles 17

M. FACE Lifecycle Management 17

N. FACE Artifact Parsing Guide 17

O. FACE Property Set 17

# AADL Annex for the FACE™ Technical Standard, Edition 3.0

Version 0.1.0, 2018-02-09

# Typography Conventions

|  |
| --- |
| Regular Text |
| AADL Keyword |
| ***FACE Keyword Introduction*** |
| **FACE Keyword** |

# 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]](#footnote-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]](#footnote-2) into AADL so that behavior and timing properties can be added and analyzed.
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 FACE architectures. Using common properties and component representations in AADL makes AADL models of FACE components portable and increases the utility of tools that operate on such AADL models.

# Background and Assumptions

1. This document provides a mapping for FACE Technical Standard Edition 3.0 and AADL 2.2.
2. 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 guidelines.
	1. As illustrated in Figure 2, the FACE Technical Standard is divided into layers. Applications live in the *Portable Components Segment* (**PCS**). Services live in the *Platform Specific Services Segment* (**PSSS**). Individual applications or services in the **PCS** or **PSSS** layer are called *Units of Portability* (**UoPs**). **UoPs** communicate with one another using the *Transport Service* (**TS**).
	2. Communication between UoPs is message based. In the FACE Technical Standard, messages are called ***views*** and are constructed from the FACE data model using ***queries***.
	3. 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.
	4. 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 **UoPs** that do not use precisely the same data representation (e.g., metric versus imperial) can have their messages translated automatically through inspection of the data model.
		1. For further information about the FACE **Data Architecture**, see section 2.3 of the FACE Technical Standard.
3. The FACE Technical Standard is designed to align with the ARINC653 standard (See section A.6 of the FACE Technical Standard). This document in turn is designed to align with the ARINC653 AADL Annex (AS5506).
	1. The ARINC653 AADL annex instructs modelers to represent ARINC653 partitions as combinations of AADL virtual processors and AADL processes. 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 AADL modeling norms.
4. 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.
5. 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.
	1. 

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

1. All communication between the FACE **PCS** and **PSSS** is conducted via the **TSS** using **Views** defined in the **Data Model** (as shown in the top and right of Figure 2).
2. In addition to its data modeling approach to interoperability of services (the FACE **PSSS**) and applications (the FACE **PCS**), 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.
	1. 

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

1. The terms specific to the FACE Technical Standard used in this annex are defined below:
	1. **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.
	2. **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.
	3. **Data Architecture Model:** The whole of Figure 1 describes the contents of the Data Architecture Model. “Data Model” is often conflated with “Data Architecture Model.” The former is a subset of the latter.
		1. Each system of integrated FACE conformant UoCs will ultimately have one **Data Model**, likely created from multiple input data models.
	4. **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.
		1. **Example**: “Altitude” is conceptual, “meters above sea level” is logical, and “32bit unsigned integer” is platform.
		2. Since **logical** and **conceptual** issues have already been addressed, the data size (in bytes) used at the **platform** level is the primary property of interest.
	5. **UoP Model**: A description of the **UoPs** in a given system of FACE conformant software and their associated views and connections. See middle of Figure 1.
		1. The connections described in the **UoP Model** do not describe inter-**UoP** communication. They provide only the **UoPs** expectations of the type of connection it will have when integrated (e.g., sampling).
		2. An integrator will combine multiple **UoP Models** (one for each integrated **UoP**) into their integrated **UoP Model**.
	6. **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.
	7. **View**: A FACE view is a message that can be passed between components. A view is composed of elements of a data model and is described by a query.
		1. **Example**: A view “status” might include altitude, airspeed, and ground speed.
		2. Views are defined in the **platform** layer of the **Data Model**.
		3. **Query**: A FACE **query** is an SQL-like expression describing features of the FACE data model to use in a view.
		4. **UoP (Unit of Portability)**: Another term for a 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.
		5. **UoPs** reside in the **PCS** or **PSSS** layers of the FACE reference architecture.
		6. Each **UoP** has an associated **USM** providing its data model definition and UoP Model definition.
	8. **UoC (Unit of Conformance)**: A DSDM or a software component designed to meet the requirements for an individual FACE segment. Units of Conformance must be verified as conformant to the FACE Technical Standard to be certified.
		1. All FACE components in the **PCS**, **TSS**, **PCS**, and **IOSS** are **UoCs**.
	9. **TSS (Transport Service Segment)**: The **TSS** is responsible for sending messages (views) between **UoPs**. The **TSS** is also responsible for translating views between **UoPs**.
		1. For example, the **TSS** might translate a “status” view to a “heartbeat” view with the same fields but different units (perhaps meters instead of feet).
		2. **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.
		3. The FACE shared data model provides common concepts such as altitude.
	10. **USM (UoP Supplied Model):** A data model provided by a software supplier that documents the data exchanged by a UoC via the TS interface. An integrated system may incorporate many **USMs**.
		1. The USM is provided as a **.**face file with each UoP.
	11. **Integrated Data Model:** The system integrator combines FACE **USMs** to create the **Integrated Data Model** for the system.
	12. **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.
	13. **System Integrator**: The system integrator is a stakeholder responsible for resolving **USMs** from FACE **UoP** vendors and for configuring the **TSS** that routes messages between **UoPs**.
	14. **FACE UUID**: Every element in the **Data Model** has a unique identifier created using the UUID standard.
	15. **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.
	16. **UoP Connection:** 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 6).
	17. **UoP EndPoint:** A **UoPEndPoint** describes the routing configuration associated with a single **UoPConnection** (See Figure 6).

# 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.
	1. Understanding of BALSA is not required to use this annex.



Figure 3: BALSA modeled in AADL

# Packaging

1. This annex does not provide specific packaging requirements. However, modelers are encouraged to create separate packages.
	1. One package for the **Data Model**
	2. One or more packages for **UoPs**
	3. One package for each **Integration Model**
2. The **USMs** for each **UoP** will 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** |  |
| 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 the TSS |  |
| integration\_model.aadl | system and system implementation for the overall system of FACE conformant components | Optionally includes time and space partitioning via process and virtual processor |

# Data Model

1. The **Data Model** (top of Figure 1) describes data relevant to a system of FACE conformant software components. The **System Integrator** uses the FACE **Shared Data Model** and **USMs** provided by **UoP** vendors to construct the **Integrated Data Model**.
	1. **UoP** vendors use or extend the **Shared Data Model**. This means that different **UoPs** will share an ontological hereditary between their views, easing the path to translating from one to the other.
2. Each entity in the **Integrated Data Model** is modeled in AADL as a data.
	1. 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::Realized\_UUID
* FACE::Realization\_Tier => logical
 |
| Data Model Entity: Platform | data or data extends… | * FACE::UUID
* FACE::Realization\_Tier => platform
* FACE::Realized\_UUID
* Memory\_Properties::Data\_Size
 |

1. Example

|  |  |
| --- | --- |
| Conceptual | data aircraftID\_Conceptual properties FACE::UUID => "{0540db6f-67fd-430c-bc72-84126daa00cc }"; FACE::Realization\_Tier => conceptual;end aircraftID\_Conceptual; |
| Logical | data aircraftID\_Logical 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; |

# Data Model Views

1. A FACE **Platform View** is composed of data from the platform tier of the FACE data model.
	1. A **Platform View’s contents are** defined by a ***query***, the semantics of which are provided in section J.3 of FACE Technical Standard Edition 3.0.
	2. A **Platform View’s** organization is defined by a template, the semantics of which are provided in section J.4 of FACE Technical Standard Edition 3.0.
	3. Each **Platform View** is modeled as a single data implementation.
	4. The subcomponents of the data implementation are determined by the **Platform View’s** **query**.
	5. The order of the subcomponents of the data implementation is determined by the **Platform View’s** **template**.

|  |  |  |
| --- | --- | --- |
| **FACE Entity** | **AADL Entity** | **Properties** |
| Conceptual View | data implementation | * FACE::UUID
* FACE::Realization\_Tier => Conceptual
 |
| Logical View | data implementation… | * FACE::UUID
* FACE::Realization\_Tier => logical
 |
| Platform View | data implementation… | * FACE::UUID
* FACE::Realization\_Tier => platform
 |
| UoP View | data implemenation | * FACE::UUID
* subcomponents
 |

1. Example

|  |  |
| --- | --- |
| Platform View | data implementation aircraft\_config.impl subcomponents aircraftID: data AircraftID\_Platform; tailNumber: data Tail\_Number\_Platform;end aircraft\_config.impl; |

# UoP Model

1. 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).
2. All AADL components translated from FACE UoCs use the FACE::UUID property to denote the UUID of the FACE component from which they were derived.
3. A collection of **UoP** **Instances** using the same TSS is modeled as the system implementation.
4. 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)

1. Each FACE UoP is modeled in AADL as a thread group.
	1. 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. In Figure 4 a single UoP is shown inside a process. However, a single process could support multiple UoPs.
	2. The ARINC653 AADL annex translates an ARINC653 partition to the combination of an AADL process and virtual processor, thus FACE UoPs must be modeled at the level of threads.
	3. 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.
	4. **UoPConnections** on the **UoP** are modeled as ports on the UoP thread group. In Figure 4 the **UoPConnection** is called ATC\_Data\_In.
	5. **UoPs** can be modeled as abstracts that are refined as thread groups. However, the use of abstract is not required and tooling may not treat an unrefined abstract as a UoP.
	6. AADL ports on **UoPs** should reference **Views** via type constraints.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 | port with data type from associated **view** | * FACE::UUID
* FACE::ViewUUID
 |  |

1. Example

|  |  |
| --- | --- |
| 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}";**end** ADSB; **thread** **group** **implementation** ADSB.impl**end** ADSB.impl; |

# TSS

1. The 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 | One abstract for the entire system implementation | * FACE::UUID
 |
| TSS (added detail) | One abstract for the entire system implementation, refined as a virtual bus | * FACE::UUID
 |
| UoP to UoP message route | connection bound to the TSS abstract.  | * FACE::UUID
 |

# 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 views (messages) 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.
	1. This document supports use of the FACE **Integration Model** as specified by the FACE Technical Standard.
	2. 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 message from one **UoP** to another).
	1. The entities of the FACE Technical Standard integration metamodel are shown in Figure 5 and Figure 6.
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**.
4. 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**.
	1. 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 6).
	2. Note that a **UoPConnection** is *not* equivalent to an AADL connection.
	3. Note that a **UoPEndPoint** is not directly equivalent to an AADL port. A **UoPEndPoint** and a **UoPConnection** together define an AADL port.
5. A **TSNodeConnection** describes the connection from a UoP to the TSS (not to another UoP)
6. A **ViewTransporter** is modeled as a connection responsible for sending views (messages) to and from UoPs.
7. A **TransportChannel** is modeled as an AADL bus to which a **ViewTransporter** connection is bound. For example, a FACE Integration Model might configure a **view** to be transported between **UoPs** by an ARINC653 Queuing Port **ViewTransporter** and adapted using a **ViewTransformation**.

|  |  |  |
| --- | --- | --- |
| **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 | subprogram call | * FACE::UUID
 |
| ViewTransporter | connection | * FACE::UUID
 |
| TransportChannel | bus with view transporter bound to it | * FACE::UUID
 |
| ViewFilter, ViewTransformation, ViewAggregation, ViewSource, ViewSink | Subprogram in a refined TSS | * FACE::UUID
 |



Figure 5 FACE Integration Package, extracted from the FACE Technical Standard Edition 3.0



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

# IOSS

1. 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.
	1. **IOSS** components are **UoCs** but not **UoPs**, as they do not use the **TSS** to communicate.
	2. **IOSS** components are modeled in AADL as abstracts.
	3. A **PSSS** UoP’s use of **IOSS** functions is modeled in AADL using subprogram calls.
	4. The physical component to which the **IOSS** service provides access is modeled in AADL as a device.
	5. 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=>IOS
 |
| IOSS Device | Device | * FACE::UUID
* FACE::Segment=>IOS
 |
| IOSS Bus | bus access | * FACE::UUID
* FACE::Segment=>IOS
 |

# FACE Health Monitoring and Fault Management (HMFM)

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

# FACE Profiles

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

# FACE Lifecycle Management

1. 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.

# FACE Artifact Parsing Guide

1. 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.

# FACE Property Set

|  |
| --- |
| **property** **set** FACE **is** Profile: **type** **enumeration** (security, safety\_extended, safety, general); Tier: **type** **enumeration** (conceptual, logical, platform); UUID: **aadlstring** **applies** **to** (**all**); Realization\_Tier: FACE::Tier **applies** **to** (**all**); segment: **type** **enumeration** (PSSS, PCS, IOSS, IOS, TSS); FaceSegment: FACE::segment **applies** **to** (**all**);**end** FACE; |

1. Unless explicitly noted, all references to the FACE Technical Standard in this document refer to Edition 3.0. [↑](#footnote-ref-1)
2. The FACE Technical Standard Edition 3.0 provides a data architecture meta model an an EMOF in section J.5. [↑](#footnote-ref-2)