



AEROSPACE STANDARD

AS5506™/4

Issued 2019-07
Reaffirmed 2025-10

Architecture Analysis & Design Language (AADL)
Annex F: AADL Annex for the FACE™ Technical Standard Edition 3.0

RATIONALE

This annex provides 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 mapping a FACE Standard Edition 3.0 Data Architecture Extensible Markup Language (XML) Metadata Interchange (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.

AS5506/4 has been reaffirmed to comply with the SAE Five-Year Review policy.

SAE Executive Standards Committee Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2025 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, or used for text and data mining, AI training, or similar technologies, without the prior written permission of SAE.

TO PLACE A DOCUMENT ORDER: Tel: 1-877-606-7323 (U.S. and Canada only)
Tel: 1-724-776-4970 (outside U.S. and Canada)
Fax: 724-776-0790
Email: CustomerService@sae.org
<http://www.sae.org>

SAE WEB ADDRESS:

Full document details: [SAE AS5506_4](#)

For more information on this standard, visit
<https://www.sae.org/standards/content/AS5506/4/>

Documents

Standard Sharing Hub

TABLE OF CONTENTS

AADL Annex for the FACE™ Technical Standard, Edition 3.0	6
Annex F.1 Scope	6
Annex F.2 Background and Assumptions	6
Annex F.3 Reference Example	9
Annex F.4 Data Model	9
Annex F.5 Data Model Views	10
Annex F.6 UoP Model	12
Annex F.7 TSS	16
Annex F.8 Routing	18
Annex F.9 IOSS	20
Annex F.10 FACE Health Monitoring and Fault Management (HMFM)	22
Annex F.11 FACE Profiles	22
Annex F.12 FACE Lifecycle Management	22
Annex F.13 FACE Artifact Parsing Guide	23
Annex F.14 FACE Property Set	23
Annex F.15 Commentary on the AADL Runtime Services	23
Annex F.16 Balsa in AADL	24
Figure 1 Architecture Segments Example, Extracted from FACE Technical Standard Edition 3.0 Section 2.4	6
Figure 2 Data Architecture, Extracted from FACE Technical Standard Edition 3.0 Section 3.9	7
Figure 3 Balsa FACE Integration Model Expressed in AADL	9
Figure 4 FACE UoP Metamodel, Extracted from the FACE Technical Standard Edition 3.0 Section J.2.6	12
Figure 5 Example UoP (ADSB) Shown Inside a Process (ADSB_process.impl)	13
Figure 6 FACE UoP Connections, Extracted from the FACE Technical Standard Edition 3.0 Section J.2.6	15
Figure 7 Balsa AADL Model with Abstract TSS Components	17
Figure 8 Balsa AADL Model with Different Concrete TSS Components	18
Figure 9 FACE Integration Package, Extracted from the FACE Technical Standard Edition 3.0 Section J.2.7	20
Figure 10 FACE Integration Transport Package, Extracted from the FACE Technical Standard Edition 3.0 Section J.2.7	20
Figure 11 IOSS Example Diagram	21
Figure 12 Notional Combined use of AADL Runtime Services and a FACE TSS Library	24
Table 1 FACE Data Model to AADL Mapping	9
Table 2 AADL Examples for the FACE Data Model	10
Table 3 Query and Template to AADL Mapping	11
Table 4 Example Platform Entity, Template, and Query	11
Table 5 Example Platform View in AADL	12
Table 6 UoP to AADL Mapping	14
Table 7 Example UoP in AADL	16
Table 8 TSS to AADL Mapping	17
Table 9 Example UoP Routing through a TSS	19
Table 10 Integration Model to AADL Mapping	19
Table 11 IOSS to AADL Mapping	21
Table 12 IOSS AADL Example	21
Table 13 AADL Property Set for the FACE Technical Standard Edition 3.0	23
Table 14 Example Balsa PCS UoPs Modeled in AADL	24
Table 15 Example Balsa PSSS UoPs Modeled in AADL	25
Table 16 Example Balsa TSS Variations Modeled in AADL	28
Table 17 Example Balsa Data Model Modeled in AADL	33

FOREWORD

- (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 3.0¹. 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 Extensible Markup Language (XML) Metadata Interchange (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 generation². 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 Essential Meta-Object Facility (EMOF) in section J.5.

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.

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 (OSATE). 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

AADL Annex for the FACE™ Technical Standard, Edition 3.0

Version 0.5.2, 2019-4-10

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 between FACE Technical Standard Edition 3.0 and AADL 2.2.
 - Unless explicitly stated otherwise, all mappings described in this annex are bi-directional.
- (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 through Annex F.12 provide mappings between FACE Technical Standard elements and AADL components.
 - Annex F.13 provides recommendations for parsing FACE data model files.
 - Annex F.14 provides the AADL property set for models of FACE aligned components.
 - Annex F.15 provides guidance on the relationship between the FACE Technical Standard and the AADL runtime services.
 - Annex F.16 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.

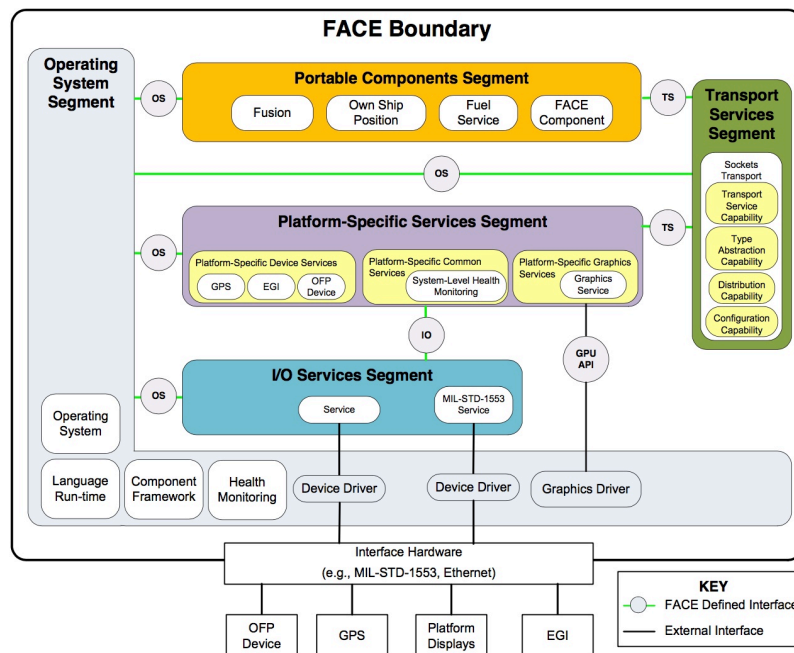


Figure 1 Architecture Segments Example, Extracted from FACE Technical Standard Edition 3.0 Section 2.4

- As illustrated in Figure 1, the FACE Technical Standard reference architecture 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.
 - 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 that two UoPs 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.
 - 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.

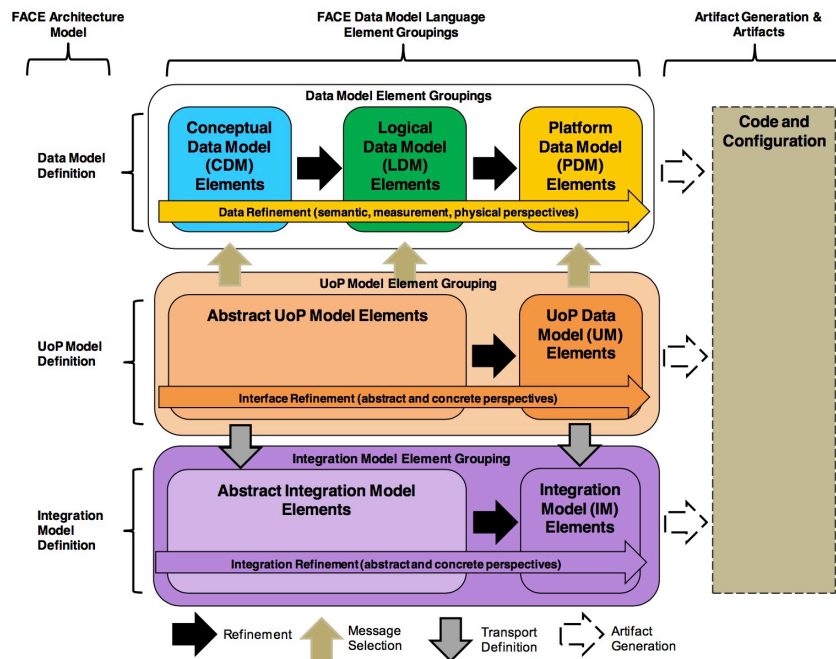


Figure 2 Data Architecture, Extracted from FACE Technical Standard Edition 3.0 Section 3.9

- The FACE Technical Standard data model provides a realization hierarchy for multiple levels of data description (*conceptual, logical, and platform*).
- 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).
- 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.
- 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 Technical 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.

- **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 conceptual, “Degrees Celsius” is logical, and “32bit unsigned integer” is platform.
- **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 a Structured Query Language (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 4 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 single 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.

- **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 Universally Unique Identifier (UUID).
 - UUIDs allow the **Integrator of FACE Conformance 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.
 - This annex does not require a specific UUID implementation.⁴
- **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.
 - A **UoPConnection** does not identify the sender or receiver on the other end of the connection (See Figure 10).
 - There are several implementations of **UoPConnection**, all of which are enumerated in section Annex F.6.
- **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.

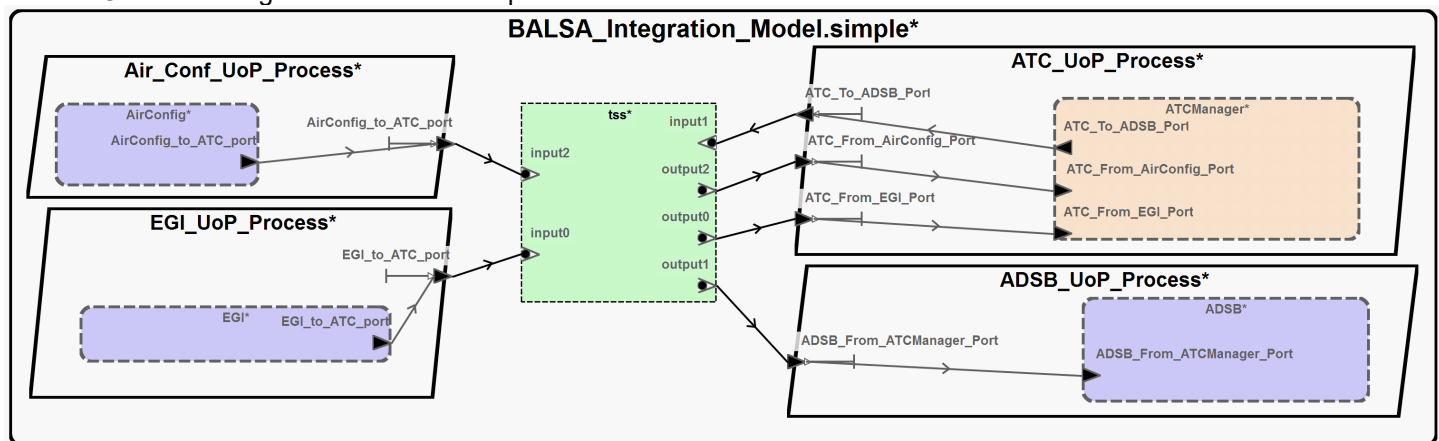


Figure 3: BALSA FACE Integration Model Expressed in AADL

Annex F.4 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 mapped to an AADL 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 1 and Table 2.

Table 1 FACE Data Model to AADL Mapping

FACE Entity	AADL Entity	Properties
Data Model	package (optional)	
Data Model Entity Composition: Conceptual	data	- FACE::UUID - FACE::Realization_Tier => conceptual

⁴ The FACE Technical Standard does not reference a specific UUID standard. An example UUID specification can be found at <http://www.ietf.org/rfc/rfc4122.txt>.

FACE Entity	AADL Entity	Properties
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

(3) Example

Table 2 AADL Examples for the FACE Data Model

Conceptual	<pre>data Aircraft_Conceptual properties FACE::Realization_Tier => conceptual; FACE::UUID => "_hwTavEM1FeiBlKadCQCZ8Q"; end Aircraft_Conceptual;</pre>
Logical	<pre>data Aircraft_Logical extends Aircraft_Conceptual properties FACE::Realization_Tier => logical; FACE::UUID => "_hwTazkM1FeiBlKadCQCZ8Q"; end Aircraft_Logical;</pre>
Platform	<pre>data Aircraft_Platform extends Aircraft_Logical properties FACE::Realization_Tier => platform; FACE::UUID => "_hwTh4EM1FeiBlKadCQCZ8Q"; end Aircraft_Platform; data AircraftID_Platform properties FACE::Realization_Tier => platform; FACE::UUID => "_hwTh10M1FeiBlKadCQCZ8Q"; end AircraftID_Platform;</pre>

Annex F.5 Data Model Views

- (1) A FACE **Conceptual View** is either a **Conceptual Query** or a **Conceptual Composite Query**.
 - Each **conceptual query** is mapped to a single data.
 - Each **conceptual composite query** is mapped to 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 mapped to a single data. If the **logical query** realizes a **conceptual query**, that realization is mapped to a data extension.
 - Each **Logical Composite Query** is mapped to a data and a data implementation. If the **logical composite query** realizes a **conceptual composite query**, that realization is mapped to 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 mapped to a single data. If the **platform template's bound query** realizes a **logical query**, that realization is mapped to a data extension.

- Each **platform composite template** is mapped to a `data` and a `data` implementation. If the **platform composite template** realizes a **logical composite query**, that realization is mapped to 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).

Table 3 Query and Template to AADL Mapping

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

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

- Note that in Table 5 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.

Table 4 Example Platform Entity, Template, and Query

Aircraft Entity	<pre> <element xsi:type="platform:Entity" xmi:id="_hwTh4EM1EeiBlKadCQCZ8Q" name="Aircraft" description="Aircraft Entity" 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> </pre>
-----------------	--



Aircraft Config Template	<pre> <element xsi:type="platform:Template" xmi:id="_hwTh8EM1EeiBlKadCQCZ8Q" name="Template_view_from_Aircraft_Config" specification="main (a) {a.aircraftID;a.tailNumber;} " boundQuery="_hwTh70M1EeiBlKadCQCZ8Q" /> </pre>
Aircraft Config Query	<pre> <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&#xA;from Aircraft as a" /> </pre>

Table 5 Example Platform View in AADL

Platform View	<pre> 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; </pre>
---------------	---

Annex F.6 UoP Model

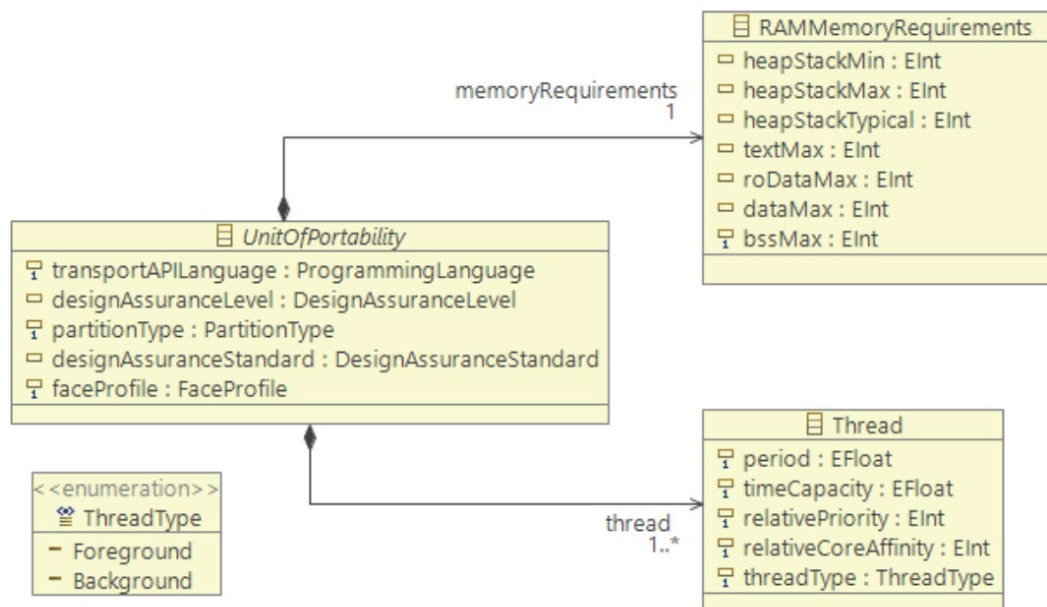


Figure 4 FACE UoP Metamodel, Extracted from the FACE Technical Standard Edition 3.0 Section J.2.6

- (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) All FACE components translated from AADL components use the `FACE::UUID` property value from the AADL component as their UUID.
- Use of this UUID enables traceability back to the original AADL component from which the FACE component was generated.
 - If an AADL component from which a FACE component is generated does not have a `FACE::UUID` property value, a new UUID should be generated for the FACE component.⁵
- (8) A collection of **UoP instances** is mapped to a `system implementation`.
- (9) The **UoP model** does not include routing of connections between **UoPs**. Connection routing is described in the **FACE integration model**.

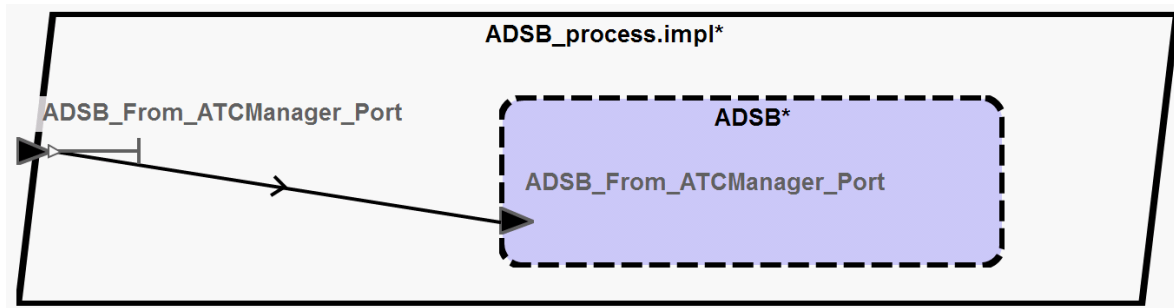


Figure 5: Example UoP (ADSB) Shown Inside a Process (ADSB_process.impl)

- (10) Each FACE **UoP** is mapped to an AADL `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**.⁶
 - Each FACE **Abstract UoP** is mapped to an AADL `abstract`.
 - i) The realization of an **Abstract UoP** by a **UoP** is mapped to an AADL `extends` specification.
 - Each **UoP** is mapped to a `thread group`.
 - A single-threaded **UoP** is mapped to 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 mapped to a `thread group` containing multiple `threads`. The FACE Technical Standard metamodel for **UoPs** is shown in Figure 4.
 - **UoPConnections** on the **UoP** are mapped to `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 4) would use an AADL `port` of type `Template_view_from_Aircraft_Config` as defined by the AADL `data` in Table 5.
 - 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 mapped to an `in event data port` and an `out event data port`.⁷
 - ii) A **QueuingConnection** is mapped to an `in event data port` or an `out event data port`.
 - iii) **SingleInstanceMessageConnection** is mapped to an `in data port` or an `out data port`.
 - The abstract **UoPConnection** is mapped to an AADL `feature` or an AADL `feature group`.
 - i) When translating from AADL to a FACE data model, concrete connection types will need to be selected for AADL `features` and `feature groups`.
 - ii) Individual AADL `features` can be mapped to either **QueuingConnection** or **SingleInstanceMessageConnection**.

⁵ Several tools provide the capability to generate UUIDs, such as the `java.util.UUID` core Java package.

⁶ 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.

⁷ 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.

- iii) A feature group containing a single in feature and a single out feature can be mapped to a **ClientServerConnection**.
- iv) Note that an AADL feature can be refined to an AADL port using the `refined to` AADL keyword.

Table 6 UoP to AADL Mapping

FACE Entity	AADL Entity	Properties	Notes
UoP	thread group	- FACE::UUID - FACE::Segment => PSSS or PCS - FACE::Profile	Can also be mapped to an abstract, but thread group is preferred.
Abstract UoP	abstract	FACE::UUID	Can also be mapped to a thread group, but abstract 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	feature or feature group	- 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, OutboundMessage corresponds to an out port.	Communication_Properties::Queue_Size set from the queuing connection's depth	