AADL Security Annex

CONTENT DRAFT

Dave Gluch

June

2019

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-0649

1 Table of Contents

2 List of Figures

3 List of Tables

This is a content draft of the SAE Architecture Analysis and Design Language (AADL) Security Annex (AADL-SA).

It is for review only.

RATIONALE

The AADL Security Annex (AADL-SA) supports the engineering of secure embedded (cyber-physical) systems by providing guidance for engineers in modeling and analyzing a system's security characteristics using the SAE Architecture Analysis and Design Language (AADL).

This Architecture Analysis & Design Language (AADL) Security Annex document was prepared by the SAE AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, Aerospace Avionics Systems Division.

A.1 Scope

(1) This document describes the Architecture Analysis & Design Language Security Annex. The AADL Security Annex provides guidance and support for specifying, modeling, and analyzing a system's security characteristics within AADL architecture models. The approach presented in this document enables the engineering of secure cyber-physical systems using the AADL and supporting tool environments such as the Open Source AADL Tool Environment (OSATE).

A.2 Rationale

(2) Software-reliant, safety-critical embedded (cyber-physical) systems often operate in untrusted environments. In these environments, they are exposed to potential attacks trying to exploit vulnerabilities to cause damage, harm people, or compromise or steal important (e.g. classified) information.

(3) As a result, security is a critical consideration in the development and operation of these systems. Security protections and properties must be correctly and completely specified, implemented, and validated throughout the system development lifecycle. To do so, engineers need to capture security policies and requirements and ensure their correct implementation in a software system architecture and design. This security annex is intended to support these activities by adding the requisite elements to the AADL language and tools such as the Open Source AADL Tool Environment (OSATE) and providing guidance to specify, model, and analyze the security aspects of a system architecture.

(4) Traditionally, software security has focused on coding issues. For example, buffer overflows, use of unsafe functions (e.g. legacy functions from the C standard libraries - strcpy - instead of their secure versions - strncpy), types mismatches, improper memory handling, etc. have been the root causes of a large number of security issues.

(5) There is increased interest in going beyond software 'bugs' to identifying flaws in the software architecture and design [Arce 2014]. Examples of flaws resulting in security problems include misunderstanding architecturally significant requirements, poor architectural implementation, and violation of established design principles resulting in architecture flaws. As a result, there is an effort underway to define a common Architecture Weaknesses Enumeration that builds on the Common Software Weaknesses Enumeration [CWE] [Mirakhorli 2016].

(6) Capturing a software architecture with its security characteristics using a formal notation enables security analysis and vulnerability discovery early in the life cycle. This has motivated the development of AADL-specific modeling capabilities, which capture security requirements and policies within AADL models. By employing appropriate tools, these AADL models can be used to implement and validate the implementation of security policies and features.

A.3 Approach

(7) This annex has been developed to support the modeling and analysis of system architecture security using the AADL. It provides guidelines and defines AADL security properties and elements to support modeling and analysis of security within system architectures.

(8) The annex describes practices that utilize the core capabilities of the AADL, security-specific property sets and verification methods, and specialized AADL components for use with the OSATE environment. There is flexibility for a user to modify and add to the property sets and components provided by this annex.

(9) In this annex, the Architecture-Led Incremental System Assurance (ALISA) environment of OSATE is used as an exemplar of the recommended capabilities for capturing and verifying security policies and requirements. Other comparable systems may be used, provided they include the basic support required by this annex.

(10) A core of security verification within this annex is the use of formal statements (e.g. predicates) whose validity is assessed across all or a portion of a system software architecture. As part of this annex, a set of example verification method statements using Resolute [Resolute 2014] and JAVA are provided. These can be used directly for analysis or as exemplars for the development of additional methods. As appropriate, other analysis and constraint languages can be employed.

A.4 Terminology

(11) The security annex relies on a set of key concepts and associated terminology useful in modeling and analyzing secure embedded systems. This set is adapted from a variety of sources and does not conform in its entirety to any specific security perspective or standard. Table 1 summaries key terms used in this annex, including relevant terms from safety analysis.

(12) For the purposes of this document we distinguish security from safety based upon intentionality. That is, security is the freedom from loss due to intentional (malicious) actors and safety is the freedom from loss due to both intentional and unintentional causes. In traditional security engineering, the primary causal concern is unauthorized access by external actors and the focus is on approaches to prevent unauthorized access. Traditionally, in safety engineering, primary considerations are losses from any cause, sometimes without explicit consideration of malicious intent.

Table 1: Safety and Security Terminology

Term	Definition	Comments
loss	A loss is a condition that results from events such as accidents [Leveson 2012] or the realizations of hazards [Feiler 2016] or threats	Loss can also be considered an event, as in the definition “the action or state of not having or keeping something any more” [CAMBRIDGE].

	[NIST 2016]. Loss is used in a general sense of harm (i.e. deleterious condition).	
accident	An accident is an undesired or unplanned event that results in a loss, including loss of human life or human injury, property damage, environmental pollution, mission loss, etc. Adapted from [Leveson 2012]	This definition encompasses loss from intentional as well as unintentional events.
hazard	A hazard is a system state or set of conditions that, together with a particular set of environmental conditions, will lead to a loss. Adapted from [Leveson 2012]	
hazard contributor	A hazard contributor is a state or set of conditions of a subsystem or component that is part of or adds to a hazard.	This supports decomposition of a system hazard.
vulnerability (security hazard)	A vulnerability (security hazard) is a system state or set of conditions (including security procedures, internal controls, design, or implementation) that could be exploited by an attacker (i.e. with a particular set of environmental conditions, will lead to a loss).	This is similar to the [NIST 2012] definition Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited by a threat source.
threat[1] (security threat)	A threat is a specified vulnerability plus a specification of an attacker, attacker access, and attacker capability to exploit the vulnerability (i.e. a *security hazard* with specified environmental conditions).	Success is ensured given an attacker with access and capability (i.e., addressing the 'will lead to' in the definition of hazard).
attack	An attack is an unauthorized attempt to access a system, usually with malicious intent.	
attacker	An attacker is an entity (or a coordinated set of entities) that engages in or attempts to engage in an attack.	

A.5 Overview

(13) The AADL Security Annex provides guidance and support for system architecture security specification, modeling, and analysis throughout the system lifecycle from concept to implementation through to operational system maintenance and upgrade.

(14) The security annex provides guidance and support to specify security policies and requirements, verify that security requirements satisfy security policies, and assure that there are mechanisms that enforce security policies and security requirements within a system architectural implementation.

(15) The security annex provides guidance and support for architecture specification, modeling, and analysis of security protections, including information/data protection, access control and protection, and action/command Protection.

(16) The security annex provides guidance and support for architecture specification, modeling, and analysis of security architectures, including specialized security

[1] In general use, the term threat encompasses both intentional and unintentional environmental conditions. In the case of 'unintentional' safety considerations a threat would be the specification of the hazard (system conditions) plus a description of the requisite environmental conditions.

architectures such as multiple independent levels of security (MILS) and cross domain solutions (CDS).

(17) The security annex provides guidance and support for the modeling and analysis of vulnerabilities/threats. – work in progress

(18) The security annex provides guidance and support for modeling and analyzing security Threats/Attacks. – TBD

A.6 Security Policies and Requirements

(19) From [NIST 2016] a security policy is a set of rules that governs all aspects of security-relevant system and system element (technology, machine, and human, elements) behavior. The rules can be stated at very high levels (e.g., organizational policy that defines acceptable behavior of employees in performing their mission/business functions) or at very low levels (e.g., an operating system policy that defines acceptable behavior of executing processes and use of resources by those processes).

(20) Security policies specify real-time security protection approaches and detection and recovery from intrusions and concomitant losses. Verification and assurance mechanisms assess the implementation of those policies, assuring that their implementation, as modeled, is correct and that the model meets the policies and requirements. For the purposes of this annex, these mechanisms do not assure the effectiveness, self-consistency, or validity of a security policy(ies) for a specific application.

(21) We consider security policies to be general statements about security attributes of a system and security requirements as statements that define the functions and capabilities that must exist within a system to satisfy security policies.

(22) The implementation elements of a system must satisfy security requirements and their associated policies. However, for some the distinction is arbitrary. In order to provide flexibility, we present an approach which enables a user to distinguish between policies and requirements.

(23) The procedures and artifacts described in this annex assume a systematic policy and requirements documentation and analysis approach that includes tool support. This approach involves both natural language and verifiable (formalized) requirements statements and the use of assurance cases for verification of policies and requirements.

(24) As a representative systematic policy and requirements documentation and analysis approach for the procedures, artifacts, and examples presented in this document, we use the Architecture-Led Incremental System Assurance (ALISA) workbench, which is part of the OSATE tool suite.

(25) In using the ALISA workbench, security policies and requirements are documented, verification plans and verification methods for those policies and requirements are defined, and assurance cases for those policies and requirements are created. The assurance cases are the foundation for system security verification.

A.7 Documenting Security Policies and Requirements

(26) As an exemplar for the documentation of security policy and requirements, we use the ReqSpec capabilities that are part of the ALISA workbench. Both security policies and security requirements are captured as requirements declarations within the ALISA notation.

(27) Security policies are organized into system requirement sets that are explicitly identified as security policy statements by naming the file as a policies file (e.g. *TransportAircraftSecuirtyPolicies.reqspec*) and the system requirements set within that file (e.g. *TransportAircraftSystemSecurityPolicies*). An excerpt from security policies for a transport aircraft system is shown in Table 2, which shows a single requirement (policy) statement named *Security*.

Table 2: Example System Security Policy

```
system requirements TransportAircraftSystemSecurityPolicies : "System-Wide Security
Policies"
for TransportAircraft_Generic::transportAircraft.generic

[
       description "These are the high level (system) security policies for an airtransport
aircraft."

       requirement Security:  "System Security must be provided"
       [
              description "Security protections that meets FAA aircraft security and
                           flight worthiness certification standards must be provided."

       ]
```

(28) Global requirements and global requirement sets can be used for security policies and requirements that apply to all the specified components of a system or security policies and requirements that can be applied to multiple systems.

(29) If no distinction is made between security policy and security requirement statements and only security requirements are defined for a system, the security requirements are captured as system requirements and organized into system requirement sets. An example for a global security requirement that all connections must be encrypted is shown in Table 3.

Table 3: Example Global Security Requirement

```
global requirements GlobalSecurity

[
       requirement encrypted_connections :
       "All connections in the system must be encrypted."  for connection
```

```
        [
                description "All connections have encryption."
                development stakeholder DevelopmentTeam.SecurityEngineer
        ]
]
```

A.8 Verification of Security Policies and Requirements

(30) The verification of security policies includes elements to validate an AADL architecture model against a specific security policy (e.g. checks of the communications and connections among components to ensure that there are no classified systems or entities communicating directly with unclassified entities).

(31) As an exemplar for the documentation of security policy and requirements, we use the capabilities that are part of the ALISA workbench. This include the verify capabilities (i.e. the verify notation) of ALISA (OSATE). Security verification plans, procedures, and activities are organized using the assurance case and verification capabilities within ALISA (i.e. the Alisa and Assure notations). The results are captured using the Assure notation with ALISA (OSATE).

(32) A verification plan is captured in a .verify file as shown in Table 4, where a portion of the verification plan for the security policies of the Transport Aircraft System. The plan includes claims for the policies being verified. The Security claim in Table 4 refers to the policy requirement statement 'Security' shown in Table 2 for the Transport Aircraft. The activities declaration in Table 4 identifies the verification method(s), specifically the activity 'ConfirmCertifications' uses the 'ReviewSecurityCertification' method that is declared in the method registry 'AircraftSecurityVerificationMethods.'

Table 4: Example Security Verification Plan

```
verification plan TransportAircraftSecurityPolicyVerificationPlan : "Verification Plan for a
Transport Aircraft Security Policies"
for TransportAircraftSystemSecurityPolicies
[
       description "This is the top level verification plan for the security policies for
the aircraft system."

       claim Security
       [
              activities
              ConfirmCertifications:
AircraftSecurityVerificationMethods.ReviewSecurityCertification ("certification
submissions")
       ]
```

(33) Verification methods are declared in method registry files. Portions of a method registry 'AircraftSecurityVerificationMethods' are shown in Table 5. The

'ReviewSecurityCertification' method is a manual review. The 'VerifyEncryption' method is a Resolute claim and 'JavaCheckTrusted' is a Java method.

Table 5: Example Verification Methods

```
verification methods AircraftSecurityVerificationMethods :
"These are the security verification methods for Aircraft Systems."

[
       method ReviewSecurityCertification (document: string) : "A formal review to confirm
compliance with all required security certifications"
       [
              manual FormalPanelReview
              description "Formal review by a certification panel to confirm compliance"
       ]
...

        method VerifyEncryption (component): "verify that a component has encryption
    A Resolute claim that the encryption property is true."
       [
              resolute
Security_Exposure_Util.Security_Exposure_Util_public.Resolute.Resolute.is_encrypted

           description "This confirms encryption."

       ]
...
method JavaCheckTrusted (component):
       "Use Java for isTrusted"
       [
       java
org.osate.securitylibrary.SecurityVerificationJava.bin.securityverification.SecurityClassifi
cationUtil.isTrusted()
       ]
```

(34) Both Resolute and Java methods are automated verification activities, which can be executed as part of the assurance case capabilities of ALISA. A portion of an assurance case for the transport aircraft is shown in Table 6, which references the verification plans for the transport aircraft security policies and requirements.

Table 6: Example Assurance Case

```
assurance case TransportAircraftAssuranceCase for
TransportAircraft_Generic::transportAircraft
 [

assurance plan TransportAircraftAssurancePlanGeneric for
TransportAircraft_Generic::transportAircraft.generic
[
       description "Security Assurance Plan for Transport Aircraft"
       assure TransportAircraftSecurityPolicyVerificationPlan
TransportAircraftSecurityVerificationPlan

]
```

```
]
```

A.9 Information Security Levels

(35) There is a primary information security level property and an information security caveats property. The information security level property is an **enumeration** property. The information level caveats property is an AADL string property. Both property declarations are shown in Table 7.

(36) In instance models, information security levels and associated caveats are applicable only to data instances and to their associated classifiers in declarative models. However, to enable modeling at abstract levels (i.e. without having to define individual data components or runtime environments) it is useful to apply information security levels and caveats to systems, processes, devices, and abstract components.

(37) Since the information security level and caveat properties are defined as inherit, a property association value within a component classifier is inherited by all subcomponents of instances of that classifier, which are instances of the component categories of the information security level or caveat property, unless overridden.

(38) With these modeling elements included, you can define a functional (conceptual) model where data is stored within and transferred among components, independent of the internal structure of the components or systems (i.e. complete implementations) using systems, processes and devices or creating models without defining a specific AADL category (i.e. using abstract). Thus, a complex cross domain system can be represented as discussed in the appendix: Cross Domain System Example.

Table 7: AADL Information Security Level Properties

```
-- Information Security Levels
       --
       Information_Security_Level: inherit enumeration (TopSecret, Secret, Confidential,
Unclassified)
       applies to (data, port, system, process, device, abstract);
       --
       Information_Security_Caveats: inherit aadlstring
       applies to (data, port, system, process, device, abstract);
```

(39) Information security properties are applied to AADL component categories that embody information (i.e. data) or those that can contain information either directly or indirectly as subcomponents. The abstract category is included, since it can be used in conceptual modeling and can be extended into one of the other categories listed.

(40) The semantics of information security level properties are presented in Table 8.

Table 8: Semantics of Information Security Level Properties for AADL Component Categories

When declared for a data classifier, all data instances of the classifier have the specified security level and all subcomponents of those instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. data, abstract), unless the inherited value is overridden.

> All of the *provides data access* data classifiers for a data instance must have the same information security level as the data instance. This requirement is not enforced by the AADL property semantics and must be ensured by the user.

When declared for a data instance, that instance has the specified security level and all subcomponents inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. data, abstract), unless the inherited value is overridden.

> All of the *provides data access* data classifiers for a data instance must have the same information security level as the data instance. This requirement is not enforced by the AADL property semantics and must be ensured by the user.

When declared for a data or event data port, the information level and information security caveats apply to the data buffer or queue associated with the port (i.e. the data or event data port maps to a static variable in the source text that represents the data buffer or queue). The data passing through the port has the information security levels specified. The information level and information security caveats property values of the port and the data classifier for the port must be the same.

When declared for a system classifier or instance, all of the subcomponents of the associated system instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited value is overridden.

> The data classifiers for all the associated system instances' external *data port*, *event data port* and *provides data access* features must have the same information security level as the system instance. These requirements are not enforced by the AADL property semantics and must be ensured by the user.

When declared for a process classifier or instance, all of the subcomponents of the associated system instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited value is overridden.

> The data classifiers for all the associated process instances' *data port*, *event data port* and *provides data access* features must have the same information security level as the process instance. These requirements are not enforced by the AADL property semantics and must be ensured by the user.

When declared for a device classifier or instance, all of the subcomponents of the associated system instances inherit the specified security level, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the inherited value is overridden.

> The data classifiers for all the associated device instances' *data port* and *event data port* features must have the same information security level as the device instance. These requirements are not enforced by the AADL property semantics and must be ensured by the user.

When declared for an abstract classifier, all extensions of that abstract classifier have the specified information security level consistent with the applicability and semantics of the information level security properties for the category of the extension.

Note that if a feature group is a feature of a component it is impacted, in that the features within the feature group, which have data classifiers, such as data ports, must comply with the semantics of information security levels presented in this section.

A.10 Information/Data Protection

(41) Data security classification properties are declared in the property set *SecuirtyClassificationProperties* and related enforcement and analysis properties in the property set *SecurityEnforcementProperties.*

(42) User defined classifications are declared in the property set *UserdefinedSecurityConstants.* Additional AADL modeling elements are available in the package *SecurityAnnexCustomPkg*.

A.11 Security Clearances

(43) The security clearance properties include a principal security classification (e.g. Top Secret, Secret, Confidential) and an supplemental statement (e.g. specialized authorizations or restrictions).

(44) A secondary clearance and secondary supplemental statement property is included. No assumption is made about the relationship of the *Security_Clearance* property and the *Secondary_Security_Clearance* property. These properties are **aadlstring** properties as shown in the declarations in Table 9.

(45) These properties are contained in the property set: *SecurityClassificationProperties*. This property set can be edited to allow modification of the enumerated values. For example, the listing can be changed to represent the security clearances of Canada or other nation.

Table 9: AADL Personnel Security Clearance Properties

```
--
       Security_Clearance: inherit enumeration (TopSecret, Secret, Confidential,
No_Clearance) applies to (system, device, processor, virtual processor, thread, thread
group, subprogram, subprogram group, process, abstract);
       --
       Security_Clearance_Supplement: inherit aadlstring applies to (system, device,
processor, virtual processor, thread, thread group, subprogram, subprogram group, process,
abstract);
       --
Secondary_Security_Clearance: inherit  enumeration (TopSecret, Secret, Confidential,
No_Clearance) applies to (system, device, processor, virtual processor, thread, thread
group, subprogram, subprogram group, process, abstract);
       --
       Secondary_Security_Clearance_Supplement: inherit aadlstring applies to (system,
device, processor, virtual processor, thread,  thread group, subprogram, subprogram group,
process, abstract);
```

(46) Security Clearance properties are applied to elements that can actively access and process information. The abstract category is included, since it can be extended into one of the other categories listed.

(47) Table 10 presents the semantics of security clearances.

Table 10: Semantics of Security Clearance Properties for AADL Component Categories

When security clearances are declared for a system classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

> When security clearances are declared for a system instance, that instance has the specified clearance or clearances and all subcomponents have the same clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

When security clearances are declared for a device classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. abstract subcomponents have the specified security) unless the property value is overridden.

> When security clearances are declared for a device instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

When security clearances are declared for a processor classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. virtual processor, abstract) unless the property value is overridden.

> When security clearances are declared for a processor instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

When security clearances are declared for a virtual processor classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. virtual processor, abstract) unless the property value is overridden.

> When security clearances are declared for a virtual processor instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

When security clearances are declared for a thread classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. subprogram, abstract) unless the property value is overridden.

> When security clearances are declared for a thread instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

When security clearances are declared for a thread group classifier, all thread subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for threads unless the property value is overridden.

When security clearances are declared for a subprogram classifier, all instances of that classifier have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. subprogram, abstract) unless the property value is overridden.

When security clearances are declared for a subprogram instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

When security clearances are declared for a subprogram group classifier, all subprogram subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for subprograms unless the property value is overridden.

When security clearances are declared for a process classifier, all instances have the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category (i.e. subprogram, thread, abstract) unless the property value is overridden.

When security clearances are declared for a process instance that instance has the specified clearance or clearances and all subcomponents have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the subcomponent category, unless the property value is overridden.

When security clearances are declared for an abstract classifier, all extensions of that abstract classifier have the specified clearance or clearances, consistent with the applicability and semantics of security clearance properties for the category of the extension.

A.12 Security Level Property

(48) The property *Security_Level* is used where no distinction is required between security subjects (i.e. *Security_Clearance properties*) and security objects (i.e. *Information_Secruity_Level* properties) in a modeling a system. For flexibility, we declare the *Security_Level_Caveats* property as an AADL string property. The declaration of the basic security level properties are shown in Table 11.

(49) The *Security_Level* property has the enumerated values as shown; however these can be replaced or extended by a user. As with the other security classification properties, the basic security level properties are declared as inherit, allowing inheritance through the architecture hierarchy.

(50) The semantics of the *Security_Level* properties are those described in Table 8 and Table 10 for the Information Security Level and Security Clearance properties.

Table 11: Security_Level Properties

```
-- Note: The SEI property set has a SecurityLevel property that is an integer.
--
       Security_Level: inherit enumeration (TopSecret, Secret, Confidential, Unclassified)
applies to (system, processor, virtual processor, thread, thread group, subprogram,
subprogram group, data, port, process, device, abstract);
--
       Security_Level_Caveats: inherit aadlstring applies to (system, processor, virtual
processor, thread, thread group, subprogram, subprogram group, data, port, process, device,
abstract);
```

A.13 Trusted Components

(51) A trusted component or system is one that is relied upon to a specified extent to enforce a specified security policy and one that has been verified to some defined level to warrant that trust. In essence, a trusted entity is one whose failure would break a security policy (i.e. the component or system is trusted to function as expected and to be unmodified from the expected [Wiki-Trusted]). For example the trusted components of Multiple Independent Levels of Security/Safety (MILS) from [Rushby 2008] are such that "trusted components ...depend only on very simple environments that can be provided with strong assurance."

(52) Trusted is not an inherited component. It must be specified for a component classifier or instance.

Table 12: Trusted Property

```
Trusted : aadlboolean applies to (system, process, thread, thread group, subprogram,
subprogram group, processor, virtual processor, bus, virtual bus, abstract);
```

A.14 Property Set Modification

(53) The property sets Security Classification and Security Enforcement Property sets can be edited by a user, including the addition of new properties.

(54) A principal objective of these editable property sets is to enable the modification of enumerated values. For example, a user might edit the *Security_Level* property to have High, Medium, and Low values rather than the values originally included in the property set, as shown in Table 13.

(55) With this capability, specific property analysis plugins can be developed that use the original property name in developing the analysis code but reads the user-specified values from the modified property set when executing an analysis.

Table 13: Example Property Modification

```
       Security_Level: inherit enumeration (High, Medium, Low)  applies to ( system,
processor, virtual processor, thread, thread group, subprogram, subprogram group, data,
port, process, device, abstract);
```

A.15 Analyzing Security Levels

(56) Resolute or other constraint or formal expression language as well as JAVA methods can be used to analyze security classifications for a system model.

(57) The annex includes example *Resolute* security functions and claims in the library (AADL package) *Security_Classifications_Resolute*. Excerpts from this library are shown in Table 14. A complete listing is included in the appendix.

(58) The JAVA package security verification has example JAVA classes and methods for analysis. These can be integrated into an ALISA analysis. A partial listing of these methods is included in the Appendix.

Table 14: Example Resolute Security Level Claims

```
        has_security_clearance (cp:component) <=
        ** "component " cp " has a security clearance" **
        has_property(cp, SecurityClassificationProperties::Security_Clearance)
        --
       has_top_secret_security_clearance (cp: component) <=
    ** " component " cp " has Top Secret security clearance" **
        property (cp, SecurityClassificationProperties::Security_Clearance, "No_Value") =
"TopSecret"
        --

        has_information_security_level (cp:component) <=
        ** "component " cp " has an information security level" **
        has_property(cp, SecurityClassificationProperties::Information_Security_Level)
        --
        has_top_secret_information_security_level (cp: component) <=
        ** " component " cp " has Top Secret security information security Level" **
        property (cp, SecurityClassificationProperties::Information_Security_Level,
"No_Value") = "TopSecret"
        --

       has_security_level (cp:component) <=
        ** "component " cp " has an information security level" **
        has_property(cp, SecurityClassificationProperties::Security_Level)

        has_top_secret_security_level (cp: component) <=
        ** " component " cp " has Top Secret security information security Level" **
        property (cp, SecurityClassificationProperties::Security_Level, "No_Value")
= "TopSecret"
        --

        all_subcomponents_have_security_level_or_are_trusted (cp: component) <=
        ** "all subcomponents of component " cp " have a value for_security level" **
        exists(sbx: subcomponents(cp)).(has_property (sbx,
SecurityClassificationProperties::Security_Level))
                and
        (forall(sb: subcomponents(cp)).(has_property (sb,
SecurityClassificationProperties::Security_Level))
        or
        has_property(sb,SecurityClassificationProperties::Trusted))

all_subcomponents_have_security (cp: component) <=
        ** "all subcomponents of component " cp " have a value for information security level
or security clearance" **
        exists(sbx: subcomponents(cp)).(has_property (sbx,
SecurityClassificationProperties::Information_Security_Level))
                and
        forall(sb: subcomponents(cp)).(has_property (sb,
SecurityClassificationProperties::Information_Security_Level) or
    has_property (sb,SecurityClassificationProperties::Security_Clearance))
    --
```

A.16 Encryption

(59) Encryption is declared using the *Encryption* property. The *Encryption* property is an AADL record type and includes fields that declare encryption details.

(60) The declarations for the record type property *Encryption* are shown Table 15, which is an excerpt from the *SecurityEnforcementProperties* property set. The declaration of the enumerated property fields *encryption_form*, *encryption_mode*, and encryption_algorithm include the entry *to_be_specified*, which can be used to require that the element, to which the encryption property is assigned, must have encryption defined at some point to complete the security specification for the system architecture.

Table 15: The Encryption Property

```
Encryption: inherit record (
       Description: aadlstring;
       encryption_form: enumeration (symmetric, asymmetric, hybrid,
       authenticated_encryption, no_encryption, to_be_specified);
       -- if the encryption form is hybrid both symmetric and asymmetric are used.
       encryption_mode: list of enumeration (ECB, CBC, CFB, CTR, GCM, CBC_MAC,
       other, to_be_specified);
       -- list is needed for hybrid encryption
       encryption_algorithm: list of aadlstring;  -- e.g. AES,  RSA, OTP
       -- a list is needed for hybrid encryption
       -- encryption algorithm list values must correspond to the key length list values
       padding: enumeration (OAEP, other, no_padding, to_be_specified);
       authenticated_encryption_type: enumeration (Encrypt_then_MAC, MAC_then_Encrypt,
Encrypt_and_MAC, AEAD, signcryption, no_authenticated_encryption);
       padding: enumeration (OAEP, other, no_padding, to_be_specified);
       key_length : list of Size; -- caution confirm that the value of key_length property
for the associated key classifier or instance is consistent
       -- a list is needed for hybrid encryption
       -- key length list values must correspond to the encryption algorithm list values
       key_type: SecurityEnforcementProperties::key_Classifier; -- references a classifier
       private_key: SecurityEnforcementProperties::key_Instance;  -- references an instance
       public_key: SecurityEnforcementProperties::key_Instance; -- references an instance
       single_key: SecurityEnforcementProperties::key_Instance;  -- references an instance
       ) applies to (data, port, abstract, system, bus, memory, device, processor,
                     virtual processor, virtual bus, connection, process, thread);
```

(61) A summary of the semantics of the encryption property is provided in Table 16.

Table 16: Encryption Properties Semantics

When encryption is declared for a data classifier or data instance, the data is encrypted as specified and all subcomponents have the specified encryption (unless overridden), consistent with the applicability and semantics of encryption properties for the subcomponent category (i.e. data subcomponents and abstract). When declared for a data or event data port, the specified encryption applies to the data buffer or queue associated with the port (i.e. the data or event data port maps to a static variable in the source text that represents the data buffer or queue). The data passing through the port is encrypted as specified. The *Encryption* property value of the port and the data classifier for the port must be the same.

When encryption is declared for an abstract classifier, its extensions have the specified encryption consistent with the applicability and semantics of encryption properties for the category of the extension.

When encryption is declared for a system classifier or instance, all subcomponents have the specified encryption, consistent with the applicability and semantics of encryption properties for the subcomponent category.

When encryption is declared for a bus classifier or a bus instance, encryption is provided as specified for all data transmitted through instances of the bus classifier or the bus instance and encryption applies to all subcomponents (unless overridden) of those instances, consistent with the applicability and semantics of encryption properties for the subcomponent category (i.e. virtual bus, abstract)

When encryption is declared for a virtual bus classifier or a virtual bus instance, encryption is provided as specified for all data transmitted through instances of the virtual bus classifier or the virtual bus instance and encryption applies to all subcomponents of those instances (unless overridden), consistent with the applicability and semantics of encryption properties for the subcomponent category (i.e. virtual bus, abstract).

When encryption is declared for a memory classifier or a memory instance encryption is provided as specified for all data contained in instances of the memory classifier or the memory instance and encryption applies to all subcomponents of those instances (unless overridden), consistent with the applicability and semantics of encryption properties for the subcomponent category (i.e. memory, bus, abstract).

When encryption is declared for a device classifier or a device instance encryption is provided as specified for all data contained in instances of the device classifier or the device instance and encryption applies to all subcomponents of those instances (unless overridden), consistent with the applicability and semantics of encryption properties for the subcomponent category (i.e. bus, virtual bus, data, abstract).

When encryption is declared for a processor classifier or a processor instance, encryption is provided as specified for all data within instances of the processor classifier or the processor instance and encryption applies to all subcomponents of those instances (unless overridden), consistent with the applicability and semantics of encryption properties for the subcomponent category (i.e. memory, bus, virtual processor, virtual bus, abstract).

When encryption is declared for a virtual processor classifier or a virtual processor instance, encryption is provided as specified for all data within instances of the virtual processor classifier or the virtual processor instance and encryption applies to all subcomponents of those instances (unless overridden), consistent with the applicability and semantics of encryption properties for the subcomponent category (i.e. virtual processor, virtual bus, abstract).

When encryption is declared for a connection instance the supporting transmission components must provide the specified encryption for all data transmitted through the connection. The connection encrypts and decrypts data, such that the data type at each end is the same.

Note that the encryption declarations can be specified as “to be specified” for an architecture model. That is, the *encryption_form* property field within the *encryption* property can be assigned the value *to_be_specified*, perhaps early in the development process and later a specific encryption property assignment can be made.

A.17 Authenticated Encryption (AE)

(62) Authenticated Encryption provides message confidentiality and integrity by employing an encryption algorithm with an authentication mechanism (e.g. message authentication code (MAC)).

(63) Both the Encryption property and the *Data_Authentication* properties are required to completely specify authenticated encryption.

A.18 Data Authentication

(64) The *Data_Authentication* property is an inherited record property as shown in Table 17. The data authentication property is used to declare that a data instance has the authentication as specified or that authentication is required by a connection (i.e. the data transmitted through the connection must be authenticated as specified) or that authentication is provided by a component (e.g. bus or virtual bus supporting a connection that requires data authentication) or when applied to a component that can contain data (e.g. system, device, abstract) it declares that all data contained in the component has authentication as specified.

Table 17: Data Authentication Property

```
Data_Authentication: inherit record
       (
              description: aadlstring;
              authentication_form : enumeration (MAC, MIC, signature, signcryption,
                                                 no_autentication, to_be_specified);
              authentication_algorithm: aadlstring;
              key_Length: Size; -- caution confirm that the value of key length for the
associated key classifier or instance is consistent
              hash_Length: Size; -- optional, if the message is hashed before
authentication. Does not apply to authenticated encryption.
              hash_algorithm: aadlstring;
              authentication_key: SecurityEnforcementProperties::key_Instance;  --
references an instance                          )
       applies to (data, port, abstract, system, bus, memory, device, processor, virtual
processor, virtual bus, connection);
       --
```

A.19 Encryption and Hash Key Management

(65) An encryption or hash key can be declared as an abstract or a data classifier. Either classifier can be extended. The examples in Table 18 extend the abstract key to create data classifiers.

Table 18: Abstract Component 'key'

```
-- A 'key' is defined as an abstract component.

abstract key
end key;

-- extend abstract key to data classifiers

       data symmetricKey extends key
       properties
              SecurityEnforcementProperties::keyLength => 128 bits;
       end symmetricKey;

       data publicKey extends key
       end publicKey;
```

```
data privateKey extends key
end privateKey;

data AESKey256 extends symmetricKey
       properties
              SecurityEnforcementProperties::keyLength => 256 bits;
end AESKey256;
```

(66) Encryption key-related properties are shown in Table 19. The cryptoPeriod is the length of time a key or certificate is valid. The key length is

Table 19: Encryption Key-Related Properties

```
Key_length: Size applies to (abstract, data); -- caution the key_length value for a
key instance must be consistent with the value of key_length field
-- declared in the key's associated encryption or data authentication property.
      crypto_period: Time applies to (abstract, data);
       Key_classifier: type classifier (abstract, data);
       Key_instance: type reference (data);
       text_type: enumeration (plainText, cipherText) applies to (abstract, data);
```

A.20 Key and Certificate Management

(67) Key and certificate management modeling and analysis is done using key-related properties, reference figure Table 19, and key classifiers, examples are shown in Table 18.

(68) Crypto key management encompasses the policies and procedures for creation, management, distribution, use, storage, and revocation of crypto keys and digital certificates.

(69) Key or certificate management can be modeled using an AADL annotated with requisite security properties. For example, a certificate can be modeled using a data classifier. As shown in TBD.

A.21 Analyzing Encryption (Cryptography)

(70) The annex includes example *Resolute* security functions and claims for analyzing encryption in the library (AADL package) *Secuirty_Enforcement_Resolute*. The library provides a basic set for use and exemplars for users to develop additional functions and claims. Some encryption analysis claims from the *Secuirty_Enforcement_Resolute* library are shown in Table 20.

Table 20: Encryption Functions and Claims

```
       -- Encryption
       has_encryption(el: aadl): bool =
       has_property (el, SecurityEnforcementProperties::Encryption)
```

```
      provides_encryption (cp:component): bool =
      has_property (cp, SecurityEnforcementProperties::Encryption)

      connection_requires_encryption (conn: connection) : bool =
   has_property (conn, SecurityEnforcementProperties::Encryption)

   is_encrypted (dt: data) <=
   ** "The data element " dt " is encrypted" **
      has_property (dt, SecurityEnforcementProperties::Encryption)
```

A.22 Access Control and Protection

(71) The security annex provides the capability to model, assess, and assure security access control. Access control and protection encompasses the classification of subjects (users/subjects) and objects (information containers and computing resources)—authorization, authentication, and access control management (i.e. assigning and modifying classifications).

A.23 Subject Authentication

(72) The Subject Authentication property declares that a subject (component instance) can participate or participates in authentication as specified, including authentication negotiations employing the specified authentication protocol, or that the component (e.g. a bus or virtual bus) supports the authentication specified. The declaration for the *Subject_Authentication* property is shown in Table 21.

(73) All components engaged in authentications, including negotiations, must have the same authentication property value. Each component bound to the component supporting authentication must have the same access authentication property value.

(74) The list of authentication types and authentication protocols can be extended by users to include new or specific authentication approaches employed in their systems.

Table 21: The Subject Authentication Property

```
   Subject_Authentication: record
   (
      AuthenticationAccessType: enumeration (no_authentication, single_password,
       smart_card,  ip_addr, two_factor, multi_layered, bio_metric, to_be_specified);
      AuthenticationProtocol: enumeration (no_authentication, cert_services, EAP, PAP,
      SPAP, CHAP, MS_CHAP, Radius, IAS, Kerberos, SSL, NTLM, to_be_specified) ;
      AuthenticationRole: enumeration (no_authentication, authenticator, accessor);
   )
   applies to (abstract, system, bus, memory, device, processor, virtual processor, virtual
bus);
```

A.24 Authorization

(75) Authorization establishes the access rights and actions of subjects and the details of authentication procedures and protocols.

(76) The security annex does not provide the capability to explicitly model or analyze authorization activities.

A.25 Security Domains

(77) The term domain in the context of security has a broad meaning [NIST 2016]. For this annex, a security domain is defined by a set of security rules (security policies) for users, processes, systems, and services that apply to activity within the domain and activity with similar entities in other domains. That is, a security domain is a physical or logical collection of (Subjects and Objects) that adheres to (complies with) a specific security policy(ies) that partitions a system based upon those security policies.

(78) The security domain property declares the name of the security domain and provides a description of the domain.

(79) The *Need_To_Know_Domains* property declares the security domains that a subject is allowed to engage with. These property declarations are shown in

(80) Table 22 presents security domain properties.

Table 22: Security Domain Related Properties

```
       Security_Domain: record (
              domain_name: aadlstring;
              description: aadlstring;
               )applies to (abstract, system);

-- A collection of security domains that the subject is allowed to engage with.

       Need_To_Know_Domains: list of aadlstring applies to (all);
```

A.26 Access Control Modeling

(81) Access control encompasses establishing the entities that should have access rights. Access control modeling includes the capability to specify and analyze specialized access control models (e.g. Bell-LaPadula, Biba) as well as intrusion detection and recovery models.

(82) These are modeled and analyzed, in part, using the Access Rights property and general access via ports between components.

A.27 Action and Command Protection

(83) The annex provides the capability to model, assess, and assure access control of execution of actions/commands including security kernels (e.g. seL4), Operating

system security controls, specialized operating systems, and hardware security module (HSM).

(84) These systems are modeled and analyzed using the core AADL constructs and security annex properties.

A.28 Security Architectures

(85) In modeling and analyzing specialized security architectures, the security annex utilizes AADL core modeling capabilities, mappings of security-specific abstractions onto AADL elements, and specialized security properties and constructs.

(86) The annex can be used to model and analyze specialized security architectures such as Multiple Independent Levels of Security (MILS) and Cross Domain Solutions (CDS).

A.29 Multiple Independent Levels of Security (MILS)

(87) Multiple Independent Levels of Security (MILS) architectures are modeled using core AADL constructs and security annex properties. An example is included in the appendix. --TBD

A.30 Cross Domain Solutions

(88) Cross domain solutions (CDS) architectures are modeled using core AADL constructs and security annex properties. An example is included in the appendix. --TBD

A.31 Analyzing Vulnerabilities/Threats/Attacks

(89) --TBD

Appendix B Informative Section

The section presents support material for the use of the security annex and examples of the application of the security annex. *This section is [Work in Progress].*

B.1 Security Policies and Requirements Examples

(90) This example captures a few of the security policies for an E-Enabled aircraft model that is described in [e-enabled]. The example consists of a set of security policies and requirements. This is an illustrative example that is not comprehensive and is not intended to be used to define policies for any operational aircraft.

(91) An AADL model of a generic e-enabled air transport aircraft is shown in Figure 1. Where the aircraft is represented as three major functional subsystems, aircraft control, airline information services, passenger Information and Entertainment Services as well as an aircraft airframe, which consists principally of hardware. The triad of major functional subsystems is a partitioning into three distinct security domains of the aircraft.

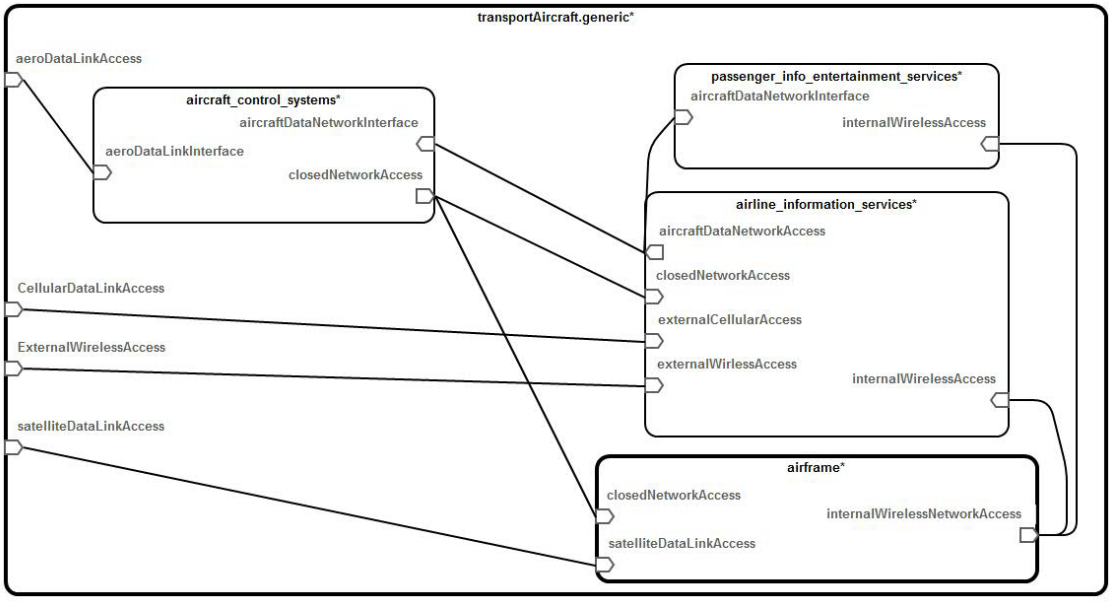


Figure 1: Generic E-Enabled Transport Aircraft

B.2 Security Policies

(92) The example consists of set of security policies that are contained in a system requirements set for the aircraft (*TransportAircraftSecuirtyPolicies.reqspec*), the developmental stakeholders for the system are from the *DevelopmentTeam* organization declared in (*Stakeholders_Development.org)*.

(93) An excerpt of security policies from the *TransportAircraftSecurityPolicies.reqspec* requirements set file is shown in Table 23.

Table 23: Example System Security Policies

```
system requirements TransportAircraftSystemSecurityPolicies : "System-Wide Security
Policies"
for TransportAircraftSystem_Generic::AirTransportOperationalSystem.multipassenger

[
   description "These are the high level (system) security policies for the
                Aircraft."

   requirement Security:  "System Security must be provided"
       [
        description "Security protections that meets FAA aircraft security and
                     flight worthiness certification standards must be provided ."

       ]

       requirement MasterSecurityPolicy: "A Master System Security Policy must be
developed and certified."
       [
       description "A master system security policy document must be developed
                    and certified by all of the agencies and organizations
                    involved in flight certification of the aircraft."
       ]

       requirement  AccessControlPolicy: "Security Controlled Access to all
                   Aircraft Systems and Resources must be provided."
                    [

         description "Access to all Aircraft operational and maintenance
                      Systems and Resources shall be permitted only by
                      authorized personnel."

         development stakeholder DevelopmentTeam.PrincipalEngineer
                                 DevelopmentTeam.SecurityEngineer
                    ]

       requirement SecureCommunicationsPolicy: "Secure communications must be
                   provided for all flight- and safety-critical systems."
                    [
          description "Communication systems must provide security measures to
                       ensure only authorized access and use."

             development stakeholder DevelopmentTeam.PrincipalEngineer
                                     DevelopmentTeam.EncryptionExpert
                    ]

]
```

B.3 Security Requirements

(94) Security requirements are captured as system requirements and organized into named system requirements sets and apply to a specific architecture system,

subsystem, or element. An example for the aircraft control system (*AircraftControl_pkg::aircraftControl.basic*) is shown in Table 24.

Table 24: Example Security Requirements Sets for Aircraft Control System

```
// Copyright 2019 Carnegie Mellon University. See Notice.txt
// Distribution Statement A: Approved for Public Release; Distribution is Unlimited.
//

system requirements securityReqs for AircraftControl_pkg::aircraftControl.basic
[
       requirement aircraftSystemsInformationSecurity: "Aircraft Systems Information
Security/Protection (ASISP) must be provided"
       [
              description "All aircraft control and flight information systems must have
security protection to ensure confidentiality, integrity, and availability."
       ]


       requirement securityAccessReq:  "Access to all aircraft data must be only by
authorized and authenticated entities"
       [
              description "All aircraft operational and performance data systems must have
security protection to ensure access only by authorized and authenticated entities."
       ]

       requirement communicationProtectionReq0: "All external and internal communications
relating to aircraft control and operation must be secure."
       [
              description "All aircraft communication systems must have security protection
to ensure access only by authorized and authenticated entities."
       ]

       requirement  communicationProtectionReq1: "All aircraft external and internal
communication for aircraft control and flight operations must employ encryption algorithms"
        [
         description "All aircraft external and internal communication for aircraft control
and flight operations must employ encryption algorithms that meet or exceed the standards
defined in NIST publication FIPS 140-2 or any superseding document that has been released.”
         ]
]
```

B.4 Security Policy Verification

(95) Verification of policies and security requirements are accomplished with the verify capabilities (i.e. the *verify* notation) of ALISA (OSATE). The procedures and activities are organized using the assurance case and verification capabilities within ALISA (i.e. the *Alisa* and *Assure* notations). The results are captured using the *Assure* notation with ALISA (OSATE).

B.5 Cross Domain System Example

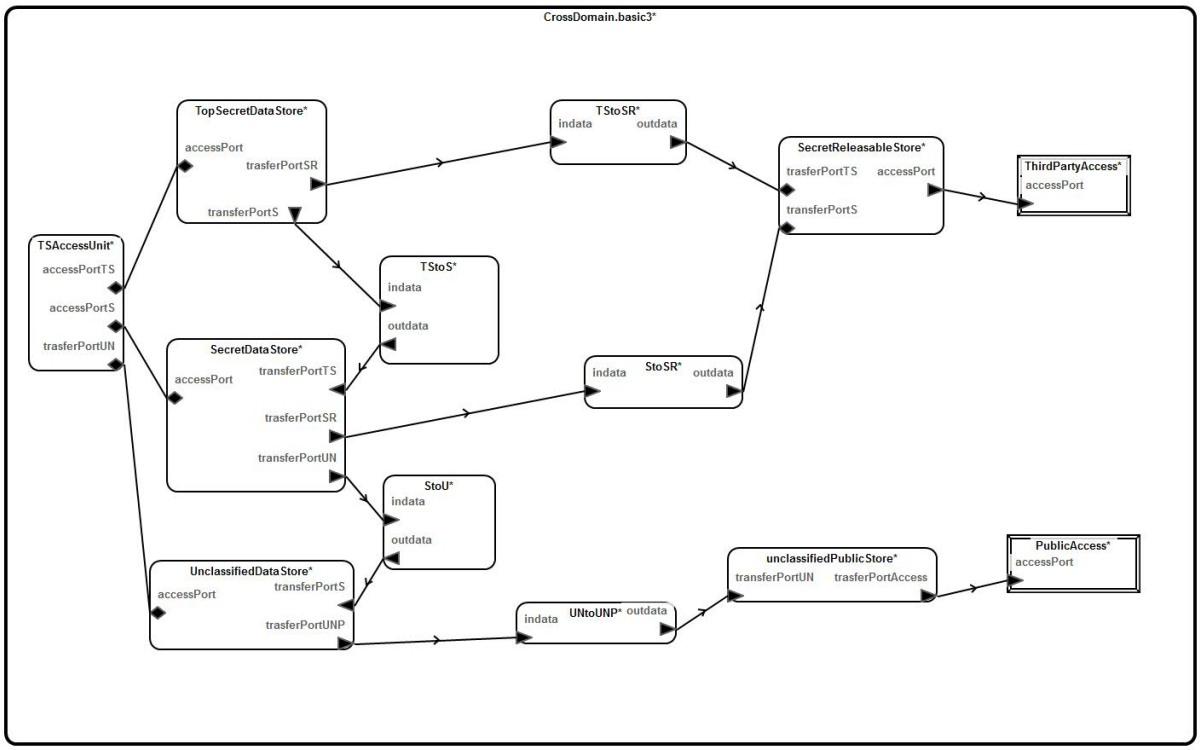
(96) This is a generalized cross domain system model that is not intended to represent any operational system.

(97) The cross domain solution (CDS) example consists of three primary data stores (top secret, secret, and unclassified) and two data stores for data that can be released (secret releasable and unclassified for public release). There is a super controller (subject) who can access and modify all three data stores (e.g. can "pump" data from a lower to a higher classification or can filter data from a higher to a lower classification) There are downgrading filters that downgrade top secret to secret, secret to unclassified, top secret to secret releasable, secret to secret releasable, and unclassified to unclassified public release. The cross domain architecture example is a blend from the references [Gehani 2012] and [Smith 2015].

(98) To simplify the example we use a single classification property Security_Level" rather than the security clearance and information security levels properties, which distinguish between subjects working with data and the objects (data). The property sets used in this work are shown in the Appendix.

(99) The AADL graphical representation of the cross domain architecture used in this work is shown in Figure 2.

Figure 2: AADL Model of the Cross Domain Architecture



(100) The three principal data stores are TopSecretStore, SecretStore, and UnclassifiedDataStore. The unclassified data store contains confidential and other data that is not for public access or use and as needed, is filtered through the filter UNtoUNP, which results in unclassified data for public release. The other filters, filter TopSecret to Secret (i.e. TStoS), TopSecret to Secret with restrictions

(TStoSR), Secret to Unclassified (StoU), and Secret to Secret with restrictions (StoSR). The data stores SecretReleaseableStore and unclassifiedPublicStore are accessible from external entities (subjects). The TSAccessUnit is the system that can access and modify all three principal data stores (e.g. can add data and “pump” data from a lower to a higher classification or can filter data from a higher to a lower classification).

(101) The TSAccessUnit and all of the filters are “Trusted” systems that can be relied upon to a specified extent to enforce a specified security policy [wiki]. Specifically, we use trusted system to mean a system that is relied upon to a specified extent to enforce a specified security policy and has been verified to some level to warrant that trust. For example the trusted components of MILS [Rushby] "trusted components … depend only on very simple environments that can be provided with strong assurance."

B.6 MILS Example

(102) This example presents the modeling and analysis of a basic multiple independent levels of security system. The system is drawn from [Rushby 2008].

B.7 AADL Property Files

(103) The security property file SecurityClassificationProperties is shown in Table 25.

Table 25: Security Classification Properties

```
-- Copyright 2019 Carnegie Mellon University. See Notice.txt
-- Distribution Statement A: Approved for Public Release; Distribution is
Unlimited.

property set SecurityClassificationProperties is

----------------------------------------
-- Security Classifications
--
	-- Security Clearances
	--
	-- The security clearance properties include a primary
	-- security classification (e.g. Top Secret, Secret, Confidential)
	-- and an optional supplemental security statement.
	--
	Security_Clearance: inherit enumeration (TopSecret, Secret, Confidential,
No_Clearance)
	applies to (system, device, processor, virtual processor, thread, thread
group, subprogram, subprogram group, process, abstract);
	--
	Security_Clearance_Supplement: inherit aadlstring
	applies to (system, device, processor, virtual processor, thread, thread
group, subprogram, subprogram group, process, abstract);
	--
	-- The secondary security clearance is provided in the event of multiple
clearances
	-- (e.g. a clearance from two different agencies.) The secondary clearance
also has a
	-- primary security classification and an optional supplemental security
statement.
	--
	-- No assumption is made about the relationship between the
Security_Clearance property
	-- and the Secondary_Security_Clearance property.
	--
	Secondary_Security_Clearance: inherit  enumeration (TopSecret, Secret,
Confidential, No_Clearance)
	applies to (system, device, processor, virtual processor, thread, thread
group, subprogram, subprogram group, process, abstract);
	--
	Secondary_Security_Clearance_Supplement: inherit aadlstring
	applies to (system, device, processor, virtual processor, thread,  thread
group, subprogram, subprogram group, process, abstract);
----------------------------------------
-- Information Security Levels
	--
	-- The information security level properties include a primary
	-- data security classification (e.g. Top Secret, Secret, Confidential)
	-- and a caveat (e.g. control markings) data security statement.
```

```
	--
	Information_Security_Level: inherit enumeration (TopSecret, Secret,
Confidential, Unclassified)
	applies to (data, port, system, process, device, abstract);
	--
    -- Information security caveats property is a string value.
	-- For example, a US information caveat value may be "//SI/TK//RELIDO"
	-- When the  security level and caveat classification are "Top Secret" and
"//SI/TK//RELIDO"
	-- this specifies the concatenated classification "TOP
SECRET//SI/TK//RELIDO"
	--
	Information_Security_Caveats: inherit aadlstring
	applies to (data, port, system, process, device, abstract);

-----------------------------------------
--  Security Levels
--
-- For models which do not need to differentiate between subject security
clearances and
-- information security levels, the SecurityLevel and SecurityLevelCaveats
properties can be used.
-- These are defined with conventional TopSecret, Secret, Confidential, and
Unclassified.
-- However, these can be changed to more generic values (e.g. High, Medium, Low) as
desired.
--
	Security_Level: inherit enumeration (TopSecret, Secret, Confidential,
Unclassified)
	applies to ( system, processor, virtual processor, thread, thread group,
subprogram, subprogram group,  data, port, process, device, abstract);
--
	Security_Level_Caveats: inherit aadlstring
	applies to (system, processor, virtual processor, thread, thread group,
subprogram, subprogram group,  data, port, process, device, abstract);

-----------------------------------------
	-- Trusted component
	--
	-- We use trusted component/system to mean a component/system that is relied
upon to a specified extent to
	-- enforce a specified security policy and has been verified to some level
to warrant that trust.
	-- For example the trusted components of MILS [Rushby] "trusted components
...
	--  depend only on very simple environments that can be provided with strong
assurance."
	--
	-- [Rushby] Rushby, J. Separation and Integration in MILS (The MILS
Constitution),
	-- Computer Science Laboratory, SRI International, Menlo Park CA 94025 USA.
	--
https://pdfs.semanticscholar.org/0398/5ca22524e10f6fab9dd966c61c4ab3de7f74.pdf
	--
```

```
      Trusted : aadlboolean applies to (system, process, thread, thread group,
subprogram, subprogram group,
            processor, virtual processor, bus, virtual bus, abstract);

-----------------------------------------
--   Security Domains
--
-- Security domain: a physical or logical collection of (Subjects and Objects) that
adheres to (complies with) a specific security policy(ies).
-- From [NIST] security domain: A domain within which behaviors, interactions, and
outcomes occur and that is defined by a governing security policy.
-- Note: A security domain is defined by rules for users, processes, systems, and
services that apply to activity within the domain and activity with similar
entities in other domains.

-- For example, a network such as NIPRNet, SIPRNet. JWICS, or NSANet with a
different security domain
-- is kept separate from other networks.
-- Systems in a domain are tagged with the same SecurityDomain value.

      Security_Domain: record (
            domain_name: aadlstring;
            description: aadlstring;
             )applies to (abstract, system);

-- The Need_To_Know_Domains property is a list of the security domains that a
subject is allowed to engage with.
-- These domains can be contain specific types of information (e.g., personnel
data) or
-- certain types of systems (e.g., flight control, mission, entertainment).

      Need_To_Know_Domains: list of aadlstring applies to (all);
--
end SecurityClassificationProperties;
```

(104) The security property file SecurityEnforcementProperties is shown in Table 26.

Table 26: Security Enforcement Properties

```
-- Copyright 2019 Carnegie Mellon University. See Notice.txt
-- Distribution Statement A: Approved for Public Release; Distribution is
Unlimited.

property set SecurityEnforcementProperties is

-------------------------------------------------
-- Data Encryption
-------------------------------------------------

      -- The encryption property is used to declare that a data instance is
encrypted as specified or that encryption is required by a connection
      -- or that encryption is provided by a component (e.g. bus or virtual bus
supporting a connection that requires encryption) or when applied
```

```
	-- to a component that can contain data (e.g. system, device, abstract) it
declares that all data contained in the component is encrypted as specified
	-- or that the component encrypts data received as specified and outputs the
encrypted data.

	Encryption: inherit record (
		Description		: aadlstring;
		encryption_form		: enumeration (symmetric, asymmetric,
hybrid, authenticated_encryption, authentication_only, no_encryption,
to_be_specified);
		-- if the encryption form is hybrid both symmetric and asymmetric are
used.
		encryption_mode		: list of enumeration (ECB, CBC, CFB, CTR,
GCM, CBC_MAC, other, to_be_specified);
		-- list is needed for hybrid encryption
		encryption_algorithm: list of aadlstring;  -- e.g. AES,  RSA, OTP
		-- a list is needed for hybrid encryption
		-- encryption algorithm list values must correspond to the key length
list values
		padding: enumeration (OAEP, other, no_padding, to_be_specified);
		authenticated_encryption_type: enumeration (Encrypt_then_MAC,
MAC_then_Encrypt, Encrypt_and_MAC, AEAD, signcryption,
no_authenticated_encryption);
		key_length		: list of Size; -- caution confirm that the value
of key_length property for the associated key classifier or instance is consistent
		-- a list is needed for hybrid encryption
		-- key length list values must correspond to the encryption algorithm
list values
		key_type		:
SecurityEnforcementProperties::key_Classifier; -- references a classifier
		private_key		: SecurityEnforcementProperties::key_Instance;  --
references an instance
		public_key		:
SecurityEnforcementProperties::key_Instance; -- references an instance
		single_key		: SecurityEnforcementProperties::key_Instance;  -
- references an instance
		) applies to (data, port, abstract, system, bus, memory, device,
processor,
					virtual processor, virtual bus, connection,
process, thread);
	--------- encryption modes-----------
	-- Electronic Code Book (ECB)
	-- Cipher block chaining (CBC)
	-- Ciphertext feedback (CFB)
	-- CTR mode (CM) or integer counter mode (ICM) or segmented integer counter
(SIC) mode
	-- Galois/Counter Mode (GCM) is a mode of operation for symmetric-key
cryptographic block ciphers
	-- Cipher block chaining (CBC)
	-- The Advanced Encryption Standard (AES) with five modes Electronic Code
Book (ECB), Cipher Block Chaining (CBC), Cipher FeedBack (CFB), Output FeedBack
(OFB), and Counter (CTR)
	--
	-------  Authenticated Encryption --------------
	--
```

```
	-- The authenticated encryption field is used to declare that a data
instance has the authenticated encryption as specified or that authenticated
encryption is required by a connection
	-- or that authenticated encryption is provided by a component (e.g. bus or
virtual bus supporting a connection that requires authenticated encryption) or when
applied
	-- to a component that can contain data (e.g. system, device, abstract) it
declares that all data contained in the component has authenticated encryption as
specified.
	--
	-- The data authentication property must be specified with the
	-- authenticated encryption field declaration to define
	-- the specifics of the authentication and encryption.
	--
	-----------------------------------------------
	-- Properties of keys for Encryption and Authentication
	-----------------------------------------------

	Key_length: Size applies to (abstract, data); -- caution the keyLength value
for a key instance must be consistent with the value
	-- of the keyLength field declared in the key's associated encryption or
data authentication property.
	-- The Crypto Period is the time span that a cryptographic key is authorized
for use
	crypto_period: Time applies to (abstract, data);
	Key_classifier: type classifier (abstract, data);
	Key_instance: type reference (data);
	text_type: enumeration (plainText, cipherText) applies to (abstract, data);

	-------------------------------
	-- Authentication Properties
	-------------------------------

	--     Data Authentication -------
	--
	-- The data authentication property is used to declare that a data instance
has authentication as specified or that authentication is required by a connection
	-- or that authentication is provided by a component (e.g. bus or virtual
bus supporting a connection that requires data authentication) or when applied
	-- to a component that can contain data (e.g. system, device, abstract) it
declares that all data contained in the component has authentication as specified.
	--
	Data_Authentication: inherit record
	(
		description              : aadlstring;
		authentication_form : enumeration (MAC, MIC, signature, signcryption,
no_autentication, to_be_specified);
		authentication_algorithm: aadlstring;
		key_Length               : Size; -- caution confirm that the value
of key length for the associated key classifier or instance is consistent
		hash_Length              : Size; -- optional, if the message is
hashed before authentication. Does not apply to authenticated encryption.
		hash_algorithm           : aadlstring;
		authentication_key  : SecurityEnforcementProperties::key_Instance;  --
references an instance
```

```
      )
      applies to (data, port, abstract, system, bus, memory, device, processor,
virtual processor, virtual bus, connection);

      --     Subject Authentication -------
      --
      --The Subject Authentication property declares that a subject (component
instance) can participate or participates in authentication as specified,
      -- including authentication negotiations employing the specified
authentication protocol, or that the component (e.g. a bus or virtual bus) supports
the authentication specified.
      --

    Subject_Authentication: record
    (
      Authentication_Access_Type: enumeration (no_authentication, single_password,
smart_card,  ip_addr, two_factor, multi_layered, bio_metric, to_be_specified);
      Authentication_Protocol: enumeration (no_authentication, cert_services, EAP,
PAP, SPAP, CHAP, MS_CHAP, Radius, IAS, Kerberos, SSL, NTLM, to_be_specified) ;
      Authentication_Role: enumeration (no_authentication, authenticator,
accessor);
    )
    applies to (abstract, system, bus, memory, device, processor, virtual
processor, virtual bus);

      --  two_factor is a subset of multi_layered but is included since it is a
prevalent multi-layered type.
      --
      --       Acronyms and values ----
      --       The Password Authentication Protocol (PAP)
      --    The Shiva PAP (SPAP)
      --    Challenge Handshake Authentication Protocol (CHAP)
      --    Microsoft CHAP (MS-CHAP)
      --    The Extensible Authentication Protocol (EAP)
      --       Remote Authentication Dial-In User Service (RADIUS)
      --       Internet Authentication Service (IAS)
      --       Secure Socket Layer (SSL)
      --       Transport Layer Security (TLS)
      --       NT (New Technology) LAN Manager (NTLM) is a suite of Microsoft
security protocols
      --       intended to provide authentication, integrity, and confidentiality
to users.
      --    Kerberos
      --    Certificate services (cert_services)

------------------------------------------
      -- certificate and encryption key management properties
------------------------------------------

      Key_distribition_method: enumeration (encrypted_channel, QKD,
direct_physical_exchange, courier, public_channel) applies to (all);

-----------------------------------------------
-- Properties related to the platform and its support for security enforcement
-----------------------------------------------
```

```
	-- name of the operating system used
	-- associated with the application component, the virtual processor, or
processor/(hardware) system
	OS: aadlstring applies to (all);

	-- name of the programming language used; associated with an application
component
	Language: aadlstring applies to (all);

	-- external exposure of component, physical or some other form
	Exposed : aadlboolean applies to (bus, virtual bus, processor, device,
system, memory);

end SecurityEnforcementProperties;
```

Table 27: Custom Security Package

```
-- Copyright 2019 Carnegie Mellon University. See Notice.txt
-- Distribution Statement A: Approved for Public Release; Distribution is
Unlimited.

-- This package contains specific component and other AADL declarations as examples
for Security modeling and analysis.


package SecurityAnnexCustomPkg

public

with SecurityEnforcementProperties;
-------------------------------------------
-------------------------------------------
-- A security 'key' is defined as an abstract component.

abstract key
end key;

-- extend abstract key to data classifiers

	data symmetricKey extends key
	properties
		SecurityEnforcementProperties::key_Length => 128 bits;
	end symmetricKey;

	data publicKey extends key
	end publicKey;

	data privateKey extends key
	end privateKey;

	data AESKey256 extends symmetricKey
		properties
```

```
                    SecurityEnforcementProperties::key_Length => 256 bits;
      end AESKey256;

-----------------------------------------
      -- certificates
      abstract CertificateAbs
      end CertificateAbs;

      data Certificate extends CertificateAbs
      end Certificate;

      data subject
      end subject;

      data implementation subject.certificate
      end subject.certificate;

      data issuer
      end issuer;

      data implementation issuer.certificate
      end issuer.certificate;

      data periodOfValidity
      end periodOfValidity;

      data implementation periodOfValidity.certificate
      end periodOfValidity.certificate;

      data AdminInformation
      end AdminInformation;

      data implementation AdminInformation.certificate
      end AdminInformation.certificate;

      data extendedInformation
      end extendedInformation;

      data implementation extendedInformation.certificate
      end extendedInformation.certificate;

      data implementation Certificate.TLS_SSL
            subcomponents
                  Subject: data subject.certificate;
                  Issuer: data issuer.certificate;
                  PeriodOfValidity: data periodOfValidity.certificate;
                  AdminInformation: data adminInformation.certificate;
                  ExtendedInformatio: data extendedInformation.certificate;
      end Certificate.TLS_SSL;


end SecurityAnnexCustomPkg;
```

4 Normative References

[DO-356]
Standard*: RTCA DO-356, Airworthiness Security Methods and Consideration*, 2014-09-23, http://standards.globalspec.com/std/9870299/rtca-do-356.

[DO-178B/C]
DO-178B/C Software Considerations in Airborne Systems and Equipment Certification, December 1992. Revised, December 2011.

[DO-254]
DO-254 Design Assurance Guidance for Airborne Electronic Hardware, April 2000.

5 Informative References

[Arce 2014]
Arce, I. et al. *AVOIDING THE TOP 10 SOFTWARE SECURITY DESIGN FL AWS*, IEEE Center for Secure Design, IEEE Computer Society, https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf.

[CNSS]
Committee on National Security Systems (CNSS) Glossary, CNSSI No. 4009, April 6, 2015

[CWE]
Common Weakness Enumeration, http://cwe.mitre.org/index.html.

[Feiler 2016]
Feiler, P. H.; Delange, J.; Gluch, D. P.; & McGregor, J.D. *Architecture-Led Safety Process*. CMU/SEI-2016-TR-012. Software Engineering institute, Carnegie Mellon University. 2016. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484826.

[Gehani 2012]
Gehani, A & Ciocarlie, G. F. Composing Cross-Domain Solutions, 2nd Layered Assurance Workshop (LAW), affiliated with the 28th Annual Computer Security Applications Conference (ACSAC), 2012, www.csl.sri.com/users/gehani/papers/LAW-2012.Streaming.pdf.

[Hansson 2008]
Hansson, J. & Feiler, P. H. Building Secure Systems using Model-Based Engineering and Architectural Models, Software Engineering Institute White Paper, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=29187.

[LAW]
The Law Dictionary, http://thelawdictionary.org/article/what-is-public-trust-security-clearance/

[Leveson 2012]
Leveson, Nancy, G. *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press. 2012.

[Mirakhorli 2016]
Mirakhorli, M. Common Architecture Weakness Enumeration (CAWE), IEEE Software Blog, April 25, 2016, http://blog.ieeesoftware.org/2016/04/common-architecture-weakness.html.

[NIST 2016]
Ross, R; McEvilley, M; and Oren, J. C. *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems*, NIST Special Publication 800-160, November 2016.

[NIST 2012]
NIST SP 800-30 Rev 1, *Guide for Conducting Risk Assessments*, National Institute of Standards and Technology, September 2012.

[OCL 2006]
Object Management Group (OMG); Object Constraint Language OMG Available Specification Version 2.0, May 2006.

[Resolute 2014]
Andrew Gacek, John Backes, Darren Cofer, Konrad Slind, & Mike Whalen, "Resolute: an assurance case language for architecture models," *Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology*, Portland, Oregon, USA, October 18 - 21, 2014, pages 19-28.

[BKCASE 2016]
BKCASE Editorial Board. 2016. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*, v. 1.7. R.D. Adcock (EIC). Hoboken, NJ:

[Rushby 2008]
Rushby, J. *Separation and Integration in MILS (The MILS Constitution)*, Technical Report SRI-CSL-08-XX, February 2008, Computer Science Laboratory, SRI International, Menlo Park CA 94025 USA. https://pdfs.semanticscholar.org/0398/5ca22524e10f6fab9dd966c61c4ab3de7f74.pdf

[Smith 2015]
Smith, Scott D. *Shedding Light on Cross Domain Solutions*, SANS Institute InfoSec Reading Room, November 6, 2015, https://www.sans.org/reading-room/.../shedding-light-cross-domain-solutions-36492.

[Shawn Fitzgerald]
Fitzgerald, Shawn, "An Introduction to Authenticated Encryption," March, 2013. https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2013/april/an-introduction-to-authenticated-encryption/ .

[Wiki-Trusted]
https://en.wikipedia.org/wiki/Trusted_system

[DND] Department of Defence Canada