What’s New in V2 of the Architecture Analysis & Design Language Standard

Peter H. Feiler
Joseph R. Seibel
Lutz Wrage

March 2010

SPECIAL REPORT
CMU/SEI-2010-SR-008

Research, Technology, and System Solutions Program
Unlimited distribution subject to the copyright.
This report was prepared for the
SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2010 Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. This document may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (http://www.sei.cmu.edu/library).
# Table of Contents

Abstract iii

1 Overview 1

2 Component Improvements 2
   2.1 New Component Categories 2
   2.2 Parameterized Component Classifiers 3
   2.3 Classifier Substitution in Refinements 3
   2.4 Component Arrays and Connection Patterns 3
   2.5 Improvements to Subprograms 4
   2.6 Improvements to Threads 5
   2.7 Thread-Related Runtime Services 5
   2.8 Asynchronous Systems 5
   2.9 Layered Architecture Modeling 6

3 Feature and Connection Improvements 7
   3.1 Abstract Features 7
   3.2 No More Refines Type 7
   3.3 Feature Groups 7
   3.4 Inverse of Feature Groups 7
   3.5 Port Queue Revisions 8
   3.6 Classifier Matching for Connections 8
   3.7 Feature Arrays and Connection Patterns 8
   3.8 Port Connections 8
   3.9 Improved Communication Timing Specification 9
   3.10 Port-Related Runtime Services 9
   3.11 Bidirectional Connections 9

4 Mode Related Improvements 10
   4.1 Mode Transition improvements 10
   4.2 Modes in Component Types 10
   4.3 Requires Modes 10
   4.4 System-Level Mode Transitions 11

5 Packages and Visibility of Classifiers 12
   5.1 With and Renames 12
   5.2 Visibility of Component Implementations 12

6 Property Improvements 14
   6.1 Specifying Applicability of Properties 14
   6.2 Revisions to Contained Property Associations 14
   6.3 Property Type Improvements 14
   6.4 Property Values 14
   6.5 References To Properties 15
   6.6 No more access keyword for properties 15
   6.7 Other Property Improvements 15

7 Other Improvements 16
   7.1 No More Anonymous Name Space 16
Abstract

This report provides an overview of changes and improvements to the Architecture Analysis & Design Language (AADL) standard for describing both the software architecture and the execution platform architectures of performance-critical, embedded, real-time systems. The standard was initially defined in the document SAE AS-5506 and published in November 2004 by SAE International (formerly the Society of Automotive Engineers). The revised language is known as AADL V2 and was published by SAE International in January 2009.

The improvements are based on feedback from actual users of the standard. Their experience and suggestions resulted in the addition of component categories for better representation of protocols as logical entities (virtual bus), scheduler hierarchies and logical time partitions (virtual processor), and a generic component (abstract). It also led to the abilities to (1) explicitly parameterize component declarations to better express architecture patterns, (2) specify multiple instances of the same component in one declaration (component array) and corresponding connection patterns, (3) set visibility rules for packages and property sets that access other packages and property sets, (4) specify system level mode transitions more precisely, and (5) use additional property capabilities including property value records.
1 Overview

The Architecture Analysis & Design Language (AADL) standard was originally published by SAE International¹ as document AS-5506 in November 2004 [SAE AS5506 2004]. The initial standard was augmented by the publication of a set of annexes containing the AADL Meta model and XMI interchange format, graphical AADL symbols, programming language interface, and the Error Model Annex [SAE AS5506/1 2006]. Based on industrial experience with the standard, corrections and improvements incorporated into AADL V2 [SAE AS5506A 2009], published in January 2009.

This report describes changes in AADL V2 as follows:

- Section 2: Component improvements, including the delineation of four additional component categories, explicit parameterization of incomplete classifier declaration, arrays of components, explicit subprogram instances and subprogram access, more flexible classifier matching and substitution, and support for layered architectures
- Section 3: Feature and connection improvements, such as more flexible connections between data and event data ports, a richer set of timing specifications for thread input and output, and expansion of port groups to support grouping of all features
- Section 4: Mode-related improvements, such as enhancement of mode transitions
- Section 5: Packages and visibility of classifiers, including improvements in the visibility of component implementations
- Section 6: Property improvements, including the availability of properties in sublanguage annexes, records as property values, and additional predeclared properties
- Section 7: Other improvements in name spaces and flows, including the elimination of the anonymous name space
- Section 8: Changes in AADL standard appendices and annexes

This report also plots the translation of models from AADL V1² to V2 (Section 9) A brief conclusion (Section 10) end the report. Throughout the report, we provide pointers to where the improvements can be found in the V2 document.

---

¹ SAE International was formerly known as the Society of Automotive Engineers (SAE).

² Throughout this report, we will refer collectively to the AADL standard version published in November 2004 and its annexes published in 2006 as V1 or AADL V1. When we refer to V2 or AADL V2, we mean the version of the standard published in January 2009, along with the revised and new annex documents planned for 2010 that are based on it.
2 Component Improvements

2.1 NEW COMPONENT CATEGORIES

Abstract component category (AADL V2 Section 4.6): Abstract components can be used to represent component models. The abstract component category can later be refined into one of the concrete component categories: software (thread, process, etc.), hardware (processor, bus, memory, device), and system. It allows for conceptual architecture modeling and later refinement into a runtime architecture. It also allows for specification of architecture patterns that can be instantiated for different component categories.

Virtual processor component category (AADL V2 Section 6.2): A virtual processor represents a logical resource that is capable of scheduling and executing threads. Virtual processors can be used to model hierarchical schedulers and recursive time partitioning of a physical processor resource. They can also be used to represent operating system threads or processes to which logical threads, active objects, or threads with the same period (rate group optimization) are bound. Virtual processors can be declared as subcomponents of processors or virtual processors, reflecting the fact that a processor is divided into logical resources. Alternatively, virtual processors can be declared separately from processors or other virtual processors and then bound to processors or virtual processors.

Virtual bus component category (AADL V2 Section 6.5): A virtual bus component represents a logical bus abstraction such as a virtual channel or communication protocol. Virtual buses can be declared as subcomponents of virtual buses, processors, and buses, or they can be declared separately and bound to virtual buses, processors, and buses. Virtual buses within a processor support connections between components on the same processor. Virtual buses on buses support connections between components across processors. Connections and virtual buses can require other virtual buses through the Allowed_Connection_Binding_Class and Allowed_Connection_Binding property. This allows users to model a protocol hierarchy. Processors and buses can specify that they provide protocols in form of virtual buses through the Provided_Virtual_Bus_Class property. Desired virtual bus quality of service (QoS) characteristics such as secure or guaranteed delivery can be specified through the Required_Connection_Quality_Of_Service property and matched with provided QoS characteristics specified by the Provided_Connection_Quality_Of_Service property.

Subprogram group component category (AADL V2 Section 5.3): Subprogram groups represent subprogram libraries. Subprogram groups can be instantiated (i.e., declared as subcomponents). Users can declare requires and provides subprogram group access and corresponding access connections. Subprograms in subprogram libraries can be referenced by subprogram calls.

---

3 Our references in parentheses to particular sections in AADL V2 can be found in document AS5506, published by SAE International [SAE AS5506A 2009]. For more information, see http://www.sae.org/technical/standards/AS5506A. Where we point to AADL V2 Annex Documents in Section 8, we indicate that the SAE AADL subcommittee plans to have these documents published in 2010.
2.2 PARAMETERIZED COMPONENT CLASSIFIERS

Prototypes (AADL V2 Section 4.7) can be used to specify classifiers as parameters for component type, component implementation, and feature group type declarations. These classifiers with prototypes can represent reference architectures, partially specified elements of a family of systems, and component templates such as a redundancy pattern for dual redundant systems. The prototypes can be referenced in place of classifiers in feature declarations, in subcomponent declarations, and as prototype bindings. This allows parameterization via prototype to be propagated down the system hierarchy.

A Prototype_Matching_Rule property can specify matching rules for actual classifiers being supplied. By default it is the classifier match. Other rules are type extension and signature match. Classifier match means they must be identical or the actual may be an implementation of the other prototype if it has only a type.

2.3 CLASSIFIER SUBSTITUTION IN REFINEMENTS

In AADL V1, classifier references in subcomponent and feature declarations could be completed in refined to statements in component type and implementation extensions. In other words, a component type could be added; or if a component type was already specified, a component implementation could be added.

In AADL V2, refined to of subcomponents (AADL V2 Section 4.5) and features (AADL V2 Section 8) also supports substitution of classifiers by extensions of those classifiers. For example, a component type can be substituted by another component type that is declared directly or indirectly by an extends of the original component type. A Classifier_Refinement_Rule property indicates whether completion or substitution is allowed. The default is classifier match.

2.4 COMPONENT ARRAYS AND CONNECTION PATTERNS

Subcomponents can be declared to be arrays (AADL V2 Section 4.5). These arrays can be single or multi-dimensional. Arrays can be declared at any level of the component hierarchy; arrays at several levels of the hierarchy effectively are cumulative from the perspective of connection patterns.

Connection patterns (AADL V2 Section 9.2.3) are applied to the source and destination component arrays of semantic connections. The pattern specifies how the semantic connection is to be replicated between the different elements of the source array and those of the destination array.

Connection patterns are specified through a Connection_Pattern property that is associated with the connection declaration. Its values determine, for each dimension of the array, how a source element in an array is connected to a destination element. Figure 1 shows the resulting connections for single-dimensional source and destination arrays of three elements. The predefined patterns are one-to-one (identity), next, cyclic next, previous, cyclic previous, all-to-all. These can be combined by listing multiple pattern values in the Connection_Pattern property.
Figure 1: One-Dimensional Array Connection Patterns

Note that the source and destination can be the same array, in which case the pattern specifies the connectivity within an array (e.g., a sensor array). Figure 2 shows examples of connection patterns for a two-dimensional array.

Figure 2: Internal Connection Pattern for a Two-Dimensional Array

If the predeclared pattern primitives and their combinations are not sufficient to express a desired connection topology, then the topology of actual connections between the elements of both arrays can be specified explicitly as lists of array index pairs through a Connection_Set property. Those values may be generated from a tool or may be the result of interpreting an algorithmic specification of the topology.

2.5 IMPROVEMENTS TO SUBPROGRAMS

In AADL V2, users can declare subprogram subcomponents to model instances of code (AADL V2 Section 5.2). The declaration of these instances is not possible in AADL V1. In AADL V2, it is optional—it is possible to model systems with subprogram instances, but it is not required to do so.
Also in AADL V2, we support explicit declaration of required and provided access to subprograms as well as subprogram access connections from calls to the subprogram instance. This supports component-based modeling where all interface requirements are documented in the interface. As in AADL V1, users can declare calls and identify the subprogram to be called by a binding property. The requires subprogram access feature replaces the server subprogram feature of AADL V1.

Subprogram calls can now also refer to subprogram instances (subcomponents directly or by referencing provides and requires subprogram access) as well as provided subprogram access in subprogram groups. Calls can now also refer to provided subprogram access in processors.

Call sequences are required to be named. Subprogram call identifiers have to be unique within the scope of the component implementation.

2.6 IMPROVEMENTS TO THREADS

Threads have two additional dispatch protocols (AADL V2 Section 5.4.2):

- **Timed**, which is an aperiodic thread that executes when an event or event data arrives or executes an alternative entry point when a timeout occurs
- **Hybrid**, which combines periodic with aperiodic (i.e., a thread that responds to both clock based dispatches and event/event data based dispatches)

Threads have additional predeclared properties, such as **Priority**. Additional service calls are available for thread-related processing, such as raising errors and retrieving error codes.

2.7 THREAD-RELATED RUNTIME SERVICES

Thread-related service calls are available to the application source code and to the runtime system generator. The API specification of these service calls has been improved to include an explicit parameter specification (AADL V2 Section 5.4.8).

2.8 ASYNCHRONOUS SYSTEMS

In a globally synchronous system, all time-related semantics are expressed in terms of a single reference time line (i.e., a single global clock). In AADL V1, the timing semantics for thread execution, communication, and mode switching are defined in terms of a globally synchronous system.

In a globally asynchronous system, there are multiple **reference times** (AADL V2 Section 5.4.7). They represent different **synchronization domains**. Any time-related coordination and communication between threads, processors, and devices across different synchronization domains must take into account differences in the reference time of each of those synchronization domains.

Processors, devices, buses, and memory can be assigned different reference times through the **Reference_Time** property. The reference time for thread execution is determined by the reference time of the processor the thread executes on. The reference time of communication between threads, devices, and processors is determined by the reference time of the source and destination, and the reference time of any execution platform component involved in the communication if it is time-driven. An application may go to a time server to retrieve time for time stamping data. This is expressed by associating a reference time directly to the application component, or the time server is explicitly modeled as part of the application.
2.9 LAYERED ARCHITECTURE MODELING

In AADL V2, a section has been added to address modeling of layered architectures (AADL V2 Section 14). It summarizes three options for modeling layered architectures:

1. hierarchical containment of components
2. layered use of threads for processing and services
3. layered virtual machine abstraction of the execution platform

In the latter case the realizations of these abstractions are represented by system implementations and associated with the component type or implementation declaration of processors, virtual processors, buses, virtual buses, memory, or devices. An Implemented_As property has been added to specify the association of a system implementation classifier with execution platform classifiers. For example, the realization of a device such as a digital camera may be modeled as a system implementation that consists of software, processors, memory, and CCD sensor devices and has the same interface as the device type declaration of the digital camera.
3 Feature and Connection Improvements

3.1 ABSTRACT FEATURES
AADL V2 introduces abstract features (AADL V2 Section 8.1) which represent placeholders for concrete features (i.e., ports, parameters, and the different kind of access features). Abstract features are typically used in incomplete component type declarations, especially those that play the role of a template. Component type extensions can refine abstract features into a concrete feature. Another method is to use feature prototypes (AADL V2 Section 4.7) to specify the concrete feature. These feature prototypes can be passed down the containment hierarchy.

Abstract features can be connected with feature connections (AADL V2 Section 9.1). Feature connections can also be used to connect abstract features to concrete features.

3.2 NO MORE REFINES TYPE
In AADL V1, users could specify implementation-specific property value for features by declaring feature refinements in the `refines type` subclause of a component implementation. These feature refinements were restricted to the addition of property associations with features.

The same can be expressed by contained property associations in the `properties` subclause of the component implementation and identify the feature in the `applies to` clause of the property association. For example

```
process myproc
features
  Port1: out data port signal;
end myproc;

process implementation myproc.impl1
properties
  Data_Model::Integer_Range=> 0 ..200 applies to Port1;
end myproc.impl1;
```

Therefore, we have removed the `refines type` subclause from AADL V2.

In this example we assume that the property `Integer_Range` has been defined in a property set called `Data_Model`. The declaration of such data modeling properties is being standardized in a Data Modeling Annex document for AADL.

3.3 FEATURE GROUPS
The `port group` concept of AADL V1 has been revised to allow for grouping of any features. The keyword has been changed to `feature group` (AADL V2 Section 8.2). Unlike port groups in AADL V1, feature groups in AADL V2 can be declared with a direction. If an `in` or `out` direction is specified as part of a feature group declaration, then all features inside the feature group must satisfy this direction.

3.4 INVERSE OF FEATURE GROUPS
In AADL V1, users have to declare a port group type and separately declare the inverse of the port group type. Port groups that are to be connected to each other could then be declared with
one or the other port group type such that features declared inside would have complementing
directions. This capability is still supported in AADL V2.

In AADL V2, we support declaration of feature groups indicating that it is the inverse of the
feature group type it references. This allows the user to not have to explicitly declare feature
group types that are the inverse of a feature group type (AADL V2 Section 8.2).

3.5 PORT QUEUE REVISIONS

In AADL V1, port queues are limited to in event and in event data ports of threads and devices.

Port queue characteristics continue to be declared through properties in AADL V2.

In AADL V2, port queues can now be associated with ports of enclosing components (thread
groups, processes, systems). This allows for the specification of a port queue with a thread group
or process that is serviced by multiple threads.

In AADL V2, port queues can also be associated with out ports (AADL V2 Section 8.3.3).

3.6 CLASSIFIER MATCHING FOR CONNECTIONS

AADL V1 requires the types of the connection source and connection destination to be identical.

AADL V2 allows the modeler to specify more flexible matching rules: classifier match (default
and same semantics as AADL V1), equivalence (match of independently defined types), subset
(the destination is considered to be a subset of the source type, e.g., to capture the DDS concept),
and conversion (an underlying protocol maps the source type into the destination type). The
Classifier_Matching_Rule property specifies which matching rule applies (AADL V2
Section 9.2).

3.7 FEATURE ARRAYS AND CONNECTION PATTERNS

AADL V2 introduces component arrays as short-hand for declaring a collection of subcompo-
nents. This collection of components may need to be connected to a component that acts as voter
or arbitrator of their output. This connection can be accomplished by declaring the incoming fea-
ture as a feature array. This feature array is limited to a single dimension. It can be declared for
any type of feature.

When a component array is connected to a component with a feature array (e.g., an array of com-
ponents with an out data port to a component with an array of data ports), a one-to-one pattern
will connect the output port of each component to a separate port of the receiving component.

3.8 PORT CONNECTIONS

(AADL V @ Section 9.2) Connections are now allowed from data ports to event data ports and
vice versa, between data or event data ports and data components—directly or via data access fea-
tures, or from event data ports to event ports.

Connections between data ports of periodic threads now can be of one of three kinds: immediate
(mid-frame), delayed (phase delayed), and sampled (potentially non-deterministic sampling).
Sampling is not available between periodic data ports in AADL V1; one has to use event data
ports with queue size one to get that effect.
In AADL V1, the symbol “->” is used for immediate connections and the symbol “->>” is used for delayed connections. In AADL V2, the Timing property is used instead of different connection symbols. Therefore, port connections can only use the symbol “->”. Timing is an enumeration property that applies to port connections. It can be assigned the value sampled, immediate, or delayed. The default value is sampled.

Port connection between an application component and a processor port can now be specified.

3.9 IMPROVED COMMUNICATION TIMING SPECIFICATION

Properties have been added to allow for explicit input and output time specification for ports (AADL V2 Section 8.3.2). This addition overrides the default timing semantics of input at dispatch time and output at completion for data ports. It allows specification of multiple input times and output times as well. If the input time is not dispatch then it is in terms of execution time.

Input and output time properties can also be specified for data access connections. In that case they specify the time ranges in which there occurs read access and write access to the shared data. Rate properties allow for specification of input and output rates for individual ports. They may be different from the thread execution rate (period).

3.10 PORT-RELATED RUNTIME SERVICES

Port-related service calls are available to the application source code, including calls to explicitly initiate sending and receiving data through ports and the processing of port queues (AADL V2 Section 8.3.5).

3.11 BIDIRECTIONAL CONNECTIONS

In AADL V2, connections can be bidirectional. The symbol “->” is used for unidirectional connections and the symbol “<->” is used for bidirectional connections. Feature connections, port connections, access connections, and feature group connections can be declared as bidirectional. Parameter connections must be unidirectional.
4 Mode Related Improvements

4.1 MODE TRANSITION IMPROVEMENTS

Mode transitions are now named (AADL V2 Section 12). By naming them, we can reference mode transitions in property reference values. Also, we can associate properties with them through contained property associations. Further, we can specify mode-transition-specific connections.. In AADL V1, specifying those connections is achieved by listing the source and destination modes of the transition.

In AADL V1, mode transition declarations refer to event ports to identify the events that can trigger the transition. These event ports can be incoming in the component type (i.e., external events that trigger mode transition) or outgoing for subcomponents. In AADL V2, users continue to have that capability and can refer to events whose source is the component itself (self.eventname).

In addition, AADL V2 allows mode transitions to be triggered by the arrival event of events, data, and event data (i.e., in addition to event ports, data ports and event data ports can be named in mode transitions as triggers). This enhancement is consistent with the ability in AADL V2 to connect a data port with an event data port (AADL V2 Section 3.8).

4.2 MODES IN COMPONENT TYPES

Mode declarations in the component type now document that a component has modal behavior (AADL V2 Section 4.3). These modes apply to all implementations. When modes are declared in a component type, mode declarations cannot be added to component implementations.

Mode declarations in the type also indicate whether modes may be externally observable. The effects of mode on a component’s behavior may be reflected in mode-specific property values through the in modes statement of property associations. In other words, property associations in the properties subclause can have different values for different modes.

An external component may control mode switching by sending an event to the component. Mode transition declaration in the component type documents those mode transitions that are triggered externally through event ports. Or, declaration in a component implementation can identify that a mode transition is triggered by an event from a subcomponent or from the component itself. Further, naming the port in the mode transition documents that an event from an event port affects a specific mode transition.

4.3 REQUIRES MODES

In AADL V2, components can inherit modes from the containing component. A component type may specify the inherited modes through a requires modes declaration (AADL V2 Section 12). The in modes clause of a subcomponent declaration can then be used to specify the mapping from the actual modes of the parent component to the inherited modes of the child component.
4.4 SYSTEM-LEVEL MODE TRANSITIONS

The specification of a transition between two system operation modes (SOM) has been improved (AADL V2 Section 13.6). It now provides support for specifying

- emergency transitions (i.e., transitions that must occur immediately)
- planned transitions (i.e., transitions that allow the application to reach the end of the hyper period of a critical set of periodic threads before performing the transition)

This specification also defines more precisely the execution and communication behavior during the actual mode transition of application threads that:
1. continue to execute in the old and new mode
2. get deactivated
3. get activated
4. are zombie threads (i.e., their execution has not completed yet at the time of the actual transition)
5 Packages and Visibility of Classifiers

5.1 WITH AND RENAMES

(AADL V2 Section 4.2) The with clause specifies the set of packages that are acceptable qual-
ifiers when classifiers are specified. In that situation, it limits the set of packages that can be
named to those listed in the with clause (e.g., when declaring subcomponents). With clauses are
declared for package sections and specify which property sets can be used to make use of property
types, property definitions, and property constants.

The renames clause defines a local (short) identifier as an alias for package names and qualified
component type references. This alias
• can only be used within the scope of the package it is declared in
• must be unique within the name space it is declared in
• can be used instead of the qualified classifier references

5.2 VISIBILITY OF COMPONENT IMPLEMENTATIONS

In AADL V2, a component implementation can be declared in both the public and private parts of
a package. The identity of a component implementation can be revealed in the public part of a
package, allowing it to be named in a subcomponent, while the details of the implementation are
hidden in the private part of the package. When this occurs, the component implementation in the
public part only contains property associations and, if appropriate, mode declarations (AADL V2
Section 4.2).

In the following example, the system implementations Gps.Dual and Gps.Secure are de-
clared in both the public and private parts of the package Visibility_Example. Notice that
declarations in the public part do not contain any subcomponents or connections; they only con-
tain modes.

```
package Visibility_Example
  --Assume that the classifiers Position_Type, Gps_Sender.Basic,
  --Gps_Sender.Secure, and GPS_Health_Monitor have been defined in this
  --package.
  public
    system Gps
      features
        Position: out data port Position_Type;
        Init.Done: in event port;
      end Gps;

    system implementation Gps.Dual
      modes
        Initialize: initial mode;
        Dualmode: mode;
        Mainmode: mode;
        Backupmode: mode;
      end Gps.Dual;

    system implementation Gps.Secure extends Gps.Dual
```
modes
  Securemode: mode;
  SingleSecuremode: mode;
end Gps.Secure;

private
system implementation Gps.Dual
subcomponents
  Main_Gps: process Gps_Sender.Basic in modes (Dualmode, Mainmode);
  Backup_Gps: process Gps_Sender.Basic in modes (Dualmode, Backupmode);
  Monitor: process GPS_Health_Monitor;
connections
  port Main_Gps.Position -> Position in modes (Dualmode, Mainmode);
  port Backup_Gps.Position -> Position in modes (Backupmode);
  port Backup_Gps.Position -> Main_Gps.SecondaryPosition in modes (Dualmode);
modes
  Started: Initialize -[ Init.Done ]-> Dualmode;
  Dualmode -[ Monitor.Backup.Stopped ]-> Mainmode;
  Dualmode -[ Monitor.Main.Stopped ]-> Backupmode;
  Mainmode -[ Monitor.All.Ok ]-> Dualmode;
  Backupmode -[ Monitor.All.Ok ]-> Dualmode;
end Gps.Dual;

system implementation Gps.Secure extends Gps.Dual
subcomponents
  Secure_Gps: process Gps_Sender.Secure in modes (Securemode);
connections
  port Secure_Gps.Position -> Position in modes (Securemode);
modes
  Dualmode -[ Monitor.Run_Secure ]-> Securemode;
  Securemode -[ Monitor.Run_Normal ]-> Dualmode;
  Securemode -[ Monitor.Backup.Stopped ]-> SingleSecuremode;
  SingleSecuremode -[ Monitor.Run_Normal ]-> Mainmode;
  Securemode -[ Monitor.Main.Stopped ]-> Backupmode;
end Gps.Secure;
end Visibility_Example;
6 Property Improvements

6.1 SPECIFYING APPLICABILITY OF PROPERTIES

In AADL V1, a property definition specifies, through keywords in the applies to clause, the component categories, features, flows and connections that properties could belong to. AADL V1 does not allow all named elements in an AADL model to have properties.

In AADL V2, however, the user can name classes in the AADL Meta model to indicate the applicability of properties. All named elements of a model can be owners of properties. This enhancement provides finer control and makes the property mechanism accessible to sublanguage annexes. For example, a user can specify that a property can be applied to a system type (type), system implementation (implementation), a system subcomponent, system classifier (type or implementation), or system (all of the above). One effect of this is that any named model element can now have properties.

Also in AADL V2, the user can now define properties to be applicable to entities in an annex subclause. In other words, the property mechanism of the AADL core language is now available to be used in annexes. For example, the Error Model Annex can now define the Occurrence property in a property set and restrict its applicability to error events. Applicability to entities in an annex subclause is specified by naming classes in the annex Meta model (AADL V2 Section 11.1.2).

6.2 REVISIONS TO CONTAINED PROPERTY ASSOCIATIONS

(AADL V2 Section 11.3) Contained property associations now

- allow a property value to be associated to multiple model elements (multiple model elements can be identified in the applies to clause).
- can be applied to elements of a component array by specifying the array index or a subrange.
- can be declared in component types, in addition to component implementations and with subcomponents. This allows users to associate properties with features and flow specifications that are declared in a component type or in a component type that is being extended.

6.3 PROPERTY TYPE IMPROVEMENTS

(AADL V2 Section 11.1.1) Record structures are now supported for properties: Users can define property types as records with multiple fields as property values. The values can be specified as list of values identified by field names.

The set of enumeration literals in the definition of an enumeration type is considered to be an ordered list.

6.4 PROPERTY VALUES

(AADL V2 Section 11.4) Property value expressions now include computed values by specifying a user-supplied function to calculate the value: compute(<function>).

[14] CMU/SEI-2010-SR-008
References as property values can now be declared to refer to any named element (or the core model or annex clauses) in an AADL model. For example, properties can refer to error events declared in the error model annex.

### 6.5 REFERENCES TO PROPERTIES

In AADL V1, users can specify the value of a property to be that of another property with the expression `value(propertyname)`.

In AADL V2, it is not necessary to use `value()` for that purpose. For properties that are not pre-declared, the property name is qualified by the `property set name`. Only predeclared property names could introduce ambiguity with `enumeration` and `units` literals, because they do not require qualification by `property set name`. For properties that do take `enumeration` or `units` literals as values, an identifier (following one of those keywords) is interpreted as a literal. If an `enumeration` or `units` literal has been defined with the same identifier as a predeclared property and the user wants to refer to the property instead of the literal, the property can be qualified with the `property set name` to indicate that the property is referenced (AADL V2 Section 11.4).

### 6.6 NO MORE ACCESS KEYWORD FOR PROPERTIES

Properties associated with access features no longer require the keyword `access`. In AADL V2 the applicability of properties can be limited to access features by specifying the following in the `applies to` clause of a property definition:

- `data access` (provides or requires data access)
- `access` (data access, bus access, subprogram access, or subprogram group access).

### 6.7 OTHER PROPERTY IMPROVEMENTS

- Instead of `Requires_Access` and `Provides_Access` properties, AADL V2 has an `Access_Right` property. It can be used on data components and ports.
- In AADL V2, thread entry points can be specified by naming a
  - subprogram in the source code
  - subprogram classifier
  - subprogram call sequence
- In AADL V2, all named elements of AADL models can be referenced, including elements in annexes.
- The predeclared properties have been organized into multiple property sets: deployment, thread, timing, communication, memory, programming, and modeling.
- Property sets have a `with` clause that specifies the set of property sets that are acceptable qualifiers when referencing a property type, a property definition, or a property constant. The use of `with` statements is unnecessary when referencing a predeclared property set.
7 Other Improvements

7.1 NO MORE ANONYMOUS NAME SPACE

The anonymous namespace in AADL V1 effectively provides a local workspace by allowing classifiers to be declared outside a package. Provision of workspaces was considered the responsibility of the tool environment; in AADL V2, the anonymous name space was eliminated. As a result, all component types, component implementations, and feature group types must be placed in packages.

7.2 FLOWS THROUGH SHARED DATA COMPONENTS

In AADL V1, flows can be specified for ports. In AADL V2, flow specifications are extended to accommodate flows through shared data components as well. The flow to and from shared data components (via data access) is determined by the Access_Right property and follows write and read access (AADL V2 Section 10.1).

7.3 END-TO-END FLOWS

An end-to-end flow may be specified as a composition of other end-to-end flows, where the last element of the predecessor end-to-end flow is connected with the first element of the successor end-to-end flow (AADL V2 Section 10.3).
8 AADL Standard Appendices and Annexes

The SAE AADL standard suite includes a number of standardized appendices and annexes. A collection of these for AADL V1 was published in 2006 [SAE AS5506/1 2006].

The SAE AADL committee is in the process of revising and adding annexes to be published in 2010. In this Section, we point to the updated and new versions of these annexes by their document letters.

8.1 AADL META MODEL & XML INTERCHANGE FORMAT STANDARD

The revised AADL meta model & XML interchange format standard document will be published in 2010. The model and interchange format provide a standard way of manipulating and interchanging AADL models. The AADL meta model has also been the basis of the OMG MARTE UML\(^4\) profile for AADL (AADL V2 Appendix Document E).

8.2 UML PROFILE FOR AADL VIA OMG MARTE

OMG MARTE has defined a UML profile for modeling embedded systems. The OMG MARTE document includes the specification of the AADL subset of MARTE as standardized UML profile for AADL. This profile will also be approved by the SAE AADL committee (AADL V2 Appendix Document F).

8.3 DATA MODELING ANNEX STANDARD

A data modeling annex standard document will be published in early 2010 that includes a standard set of properties and a collection of predeclared basic data component types (AADL V2 Annex Document B).

8.4 BEHAVIOR ANNEX STANDARD

A behavior annex standard document will be published in early 2010 that allows modelers to annotate component types and implementations with behavior specifications (AADL V2 Annex Document D).

8.5 ERROR MODEL ANNEX STANDARD

The error model annex standard document is being revised to support AADL V2 with publication expected in late 2010. It allows modelers to annotate component types and implementations with fault behavior specifications including probabilistic fault occurrence and propagation (AADL V2 Annex Document C)

\(^4\) OMG is the Object Management Group; MARTE stands for Modeling and Analysis of Real-time and Embedded systems; UML is the Unified Modeling Language.
8.6 CODE GENERATION ANNEX STANDARD

A code generation annex standard document is being developed that provides guidance and a standardized set of properties to support automatic generation and integration of runtimes systems and application components (AADL V2 Annex Document A).

8.7 ARINC653 ANNEX STANDARD

The ARINC653 annex standard document is being developed to provide guidance and a set properties to support modeling of partitioned architectures according to the ARINC653 standard. It is expected to be published in 2010.
9 Translation from AADL V1 to AADL V2

This Section summarizes language constructs in AADL V1 that are affected by changes in AADL V2 and thus require translation when migrating AADL models from V1 to V2.

9.1 AADL SPECIFICATIONS AND ANONYMOUS NAME SPACES

In AADL V2, all classifier declarations must be placed in packages. Thus, any component type, component implementation, port group type, and annex library declaration that is not placed in a package must be placed in a package.

Translation Action

Use the name of the file that contains such declarations as the name of the package. Since these items can only be referenced within that package there is no need for additional corrections.

\[
\text{AADL\_specification ::= }
\{ \text{AADL\_global\_declaration | AADL\_declaration} \}^+ \]

9.2 PACKAGE DECLARATIONS AND PROPERTIES

In AADL V1, both the public and the private section of a package could have a properties section. In AADL V2, there is a single properties section after the public and private section.

Translation Action

The property associations are moved into the single properties section:

\[
\text{package\_spec ::= }
\text{package defining\_package\_name }
\{ \text{public package\_declaration [ private package\_declaration ]} \\
\text{ | private package\_declaration } \\
\text{ | properties ( \{ property\_association \}^+ | none\_statement \}) \\
\text{end defining\_package\_name ;} \\
\text{package\_declaration ::= \{ aadl\_declaration \}^+} \\
\text{| properties ( \{ property\_association \}^+ | none\_statement \})^+} \]

9.3 PACKAGE DECLARATIONS AND WITH CLAUSES

In AADL V2, we have with clauses that restrict the packages that can be referenced by a given package.

Translation Action

Insert with clause for those packages that are actually referenced.
9.4 REFINES TYPE IN COMPONENT IMPLEMENTATIONS

In AADL V1, component implementations have a *refines type* section that allows users to declare property associations with implementation specific values for features in the type.

**Translation Action**

Convert those property associations into contained property associations naming the feature and place them in the properties section of the component implementation.

9.5 NAMING OF SUBPROGRAM CALL SEQUENCES

In AADL V1, call sequence naming is optional, while in AADL V2 it is required.

**Translation Action**

Add a generated name as call sequence identifier.

9.6 NAMED MODE TRANSITIONS

In AADL V1, the *in* modes clause refers to a mode transition by naming the source and destination modes. One reason is that mode transitions are not named.

In AADL V2, mode transitions can optionally have names and a reference to a mode transition must be expressed by referring to its name.

**Translation Action**

Replace the reference by source and destination mode with the name and attach an identifier to those mode transition declarations.

9.7 CHANGES FOR FEATURES

**Translation Action**

Translate the following reserved words for feature declarations in AADL V1 into reserved words in AADL V2:

<table>
<thead>
<tr>
<th>V1</th>
<th>Port group</th>
<th>Server subprogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>Feature group</td>
<td>Provides subprogram access</td>
</tr>
</tbody>
</table>

9.8 CHANGES FOR CONNECTIONS

**Translation Action**

Translate the following reserved words for connection declarations in AADL V1 into reserved words in AADL V2:

<table>
<thead>
<tr>
<th>V1</th>
<th>Data port</th>
<th>Event port</th>
<th>Event data port</th>
<th>Port group</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>Port</td>
<td>Port</td>
<td>Port</td>
<td>Feature group</td>
</tr>
</tbody>
</table>

Also, for data port connections, the connection symbol is no longer used to distinguish between immediate and delayed connections. This information is now stored in a property on the connection.
Translation Action
If the connection symbol is “->”, add the property “Timing => immediate;” to the connection. If the connection symbol is “->>”, change the symbol to “->” and add the property “Timing => delayed;” to the connection.

9.9 PROPERTY SETS AND WITH CLAUSES
In AADL V2, a \texttt{with} clause restricts references to properties in property sets other than the pre-declared properties to those listed in the \texttt{with} clause. In AADL V1, there is no such restriction.

Translation Action
Insert with clause naming all property sets that are actually referenced.

9.10 PROPERTY DEFINITION CHANGES
In AADL V1, acceptable references for the reference type and acceptable property owners (applies to) have special syntax used reserved words.
In AADL V2, these become identifiers into the AADL meta model.

Translation Action

<table>
<thead>
<tr>
<th>Translate V1</th>
<th>Into V2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For referable element categories</strong></td>
<td></td>
</tr>
<tr>
<td>Connections</td>
<td>Connection</td>
</tr>
<tr>
<td>Server subprogram</td>
<td>Subprogram access</td>
</tr>
<tr>
<td><strong>For property owner categories</strong></td>
<td></td>
</tr>
<tr>
<td>Port group</td>
<td>Feature group, feature group type</td>
</tr>
<tr>
<td>Server subprogram</td>
<td>Subprogram access</td>
</tr>
<tr>
<td>Port group connections</td>
<td>Feature group connection</td>
</tr>
<tr>
<td>Event port connections</td>
<td>Port connection</td>
</tr>
<tr>
<td>Data port connections</td>
<td>Port connection</td>
</tr>
<tr>
<td>Event data port connections</td>
<td>Port connection</td>
</tr>
<tr>
<td>Port connections</td>
<td>Port connection</td>
</tr>
<tr>
<td>Access connections</td>
<td>Access connection</td>
</tr>
<tr>
<td>Parameter connections</td>
<td>Parameter connection</td>
</tr>
</tbody>
</table>
9.11 CHANGES IN PROPERTY EXPRESSIONS

AADL V1 requires an access reserved word for some properties. In AADL V2, this reserved word is not required.

Translation Action
Remove this reserved word in property definitions and property associations.

In AADL V1 the value of another property used as property value requires a value (propertyname). In AADL V2, value is not required.

Translation Action
Remove value (propertyname);

In AADL V1, a classifier term (i.e., naming of a classifier) requires the specification of the component category (e.g., foo => system (gps));. In AADL V2, a classifier term uses classifier instead of the category (i.e., foo => classifier(gps));

Translation Action
Replace category name by “classifier”.

In AADL V1, a classifier term and a reference term do not require parentheses around the value. In AADL V2 a classifier term uses classifier (classifier name); and reference (model element path);

Translation Action
Add parentheses for classifier and reference terms.

9.12 RENAMING OF PROPERTIES

In AADL V1, we have a number of entry point properties; they end in Entrypoint. In AADL V2 these properties end in Entrypoint_Source_Text.

In AADL V1, we have Required_Access and Provided_Access as properties. In AADL V2 these are replaced by Access_Right.

In AADL V1, there is no predeclared property called Priority. Instead, this property is defined in the property set called “SEI.” In AADL V2, this property is part of the predeclared set of properties. Thus, the property set name can be left off.
10 Conclusions

The changes that have been incorporated into AADL V2 allow the standard to better meet the needs of safety-critical, embedded, real-time system engineering. The language has become more expressive in that AADL V2 is to a large extent a small superset of AADL with a few additional constructs. Also, the semantic expressiveness of existing constructs has been enriched, mostly through additional properties. As a result, models expressed in the original AADL can be mapped into models expressed in AADL V2.

Care has been taken that depreciated constructs of AADL can be mapped into AADL V2 representations. This allows for an automatic conversion of existing AADL models into AADL V2 models.
References

URLs are valid as of the publication date of this document.

[SAE AS5506 2004]

[SAE AS5506/1 2006]

[SAE AS5506A 2009]
# What’s New in V2 of the Architecture Analysis & Design Language Standard

This report provides an overview of changes and improvements to the Architecture Analysis & Design Language (AADL) standard for describing both the software architecture and the execution platform architectures of performance-critical, embedded, real-time systems. The standard was initially defined in the document SAE AS-5506 and published in November 2004 by SAE International (formerly the Society of Automotive Engineers). The revised language is known as AADL V2 and was published by SAE International in January 2009.

The improvements are based on feedback from actual users of the standard. Their experience and suggestions resulted in the addition of component categories for better representation of protocols as logical entities (virtual bus), scheduler hierarchies and logical time partitions (virtual processor), and a generic component (abstract). It also led to the abilities to (1) explicitly parameterize component declarations to better express architecture patterns, (2) specify multiple instances of the same component in one declaration (component array) and corresponding connection patterns, (3) set visibility rules for packages and property sets that access other packages and property sets, (4) specify system level mode transitions more precisely, and (5) use additional property capabilities including property value records.

### SUBJECT TERMS
- AADL, embedded systems, architectural modeling, real-time systems

### NUMBER OF PAGES
29