Software Engineering Institute

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

Peter H. Feiler
Joseph R. Seibel
Lutz Wrage

March 2012

SPECIAL REPORT
CMU/SEI-2011-SR-011

Research, Technology, and System Solutions Program

http://www.sei.cmu.edu

CarnegieMellon

Table of Contents

List of Figures

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 2004 by SAE International (formerly the Society of Automotive Engineers). The revised language, known as AADL V2, was published in 2009. Feedback from users of the standard guided the plan for improvements. Their experience and suggestions resulted in the addition of component categories to better represent protocols as logical entities (virtual bus), scheduler hierarchies and logical time partitions (virtual processor), and a generic component (abstract). Revisions 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. This project was funded by the U.S. Army Aviation and Missile Research Development and Engineering Center Software Engineering Directorate (AMRDEC SED).

1 Overview

The Architecture Analysis & Design Language (AADL) standard was originally published by SAE International[1] as document AS-5506 in November 2004 [SAE 2004]. The initial standard was augmented by the publication of a set of annexes containing the AADL Meta model and XML Metadata Interchange (XMI) format, graphical AADL symbols, programming language interface, and the Error Model Annex [SAE 2006]. SAE International incorporated corrections and improvements, based on industrial experience with the standard, into AADL V2 in January 2009 [SAE 2009]. In January 2011, SAE International published a second set of annexes consisting of the Behavior Annex, Data Modeling Annex, and ARINC653 Annex [SAE 2011].

This report describes changes in AADL V2 as follows:

- Section 2: Component improvements, including four additional component categories, explicit parameters for incomplete class 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 namespaces and flows, including eliminating the anonymous namespace
- Section 8: Changes in AADL standard appendices and annexes

This report also plots the translation of models from AADL V1 to V2 (Section 9).[2] A brief conclusion (Section 10) ends the report. Throughout the report, we provide pointers to the locations of improvements in the V2 document.

Additional resources regarding the SAE AADL standard are available at http://www.aadl.info and the public AADL Wiki [AADL Wiki 2011]. This project was funded by the U.S. Army Aviation and Missile Research Development and Engineering Center Software Engineering Directorate (AMRDEC SED).

[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 based on AADL V2 that were published in January 2011 and those in review.

2 Component Improvements

2.1 New Component Categories

Abstract component category (AADL V2 Section 4.6):[3] Abstract components can represent component models. Later, users can refine the abstract component category into one of the concrete component categories: software (thread, process, etc.), hardware (processor, bus, memory, and device), and system. Abstract components allow conceptual architecture modeling and later refinement into a runtime architecture. They also allow users to specify architecture patterns that can instantiate different component categories.

Virtual processor component category (AADL V2 Section 6.2): A virtual processor represents a logical resource that schedules and executes threads. Virtual processors model hierarchical schedulers and recursive time partitioning of a physical processor resource. They also represent operating-system threads or processes to which logical threads, active objects, or threads with the same period (e.g., rate group optimization) are bound. Users can declare virtual processors as subcomponents of processors or virtual processors, reflecting the fact that a processor is divided into logical resources. Alternatively, users can declare virtual processors separately from processors or other virtual processors and then bind them 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. Users can declare virtual buses as subcomponents of virtual buses, processors, and buses, or they can declare these components separately and then bind them 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 users provide protocols in the form of virtual buses through the `Provided_Virtual_Bus_Class` property. Users can specify desired characteristics for the quality of service (QoS) provided by the virtual bus, such as secure or guaranteed delivery through the `Required_Connection_Quality_Of_Service` property, and match them with 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. Users can instantiate subprogram groups (i.e., declare them as subcomponents). Users can declare `requires` and `provides` subprogram group access and corresponding access connections. Subprogram calls can reference subprograms in subprogram libraries.

[3] Our references in parentheses to particular sections in AADL V2 correspond to document AS5506A, published by SAE International [SAE 2009]. See http://www.sae.org/technical/standards/AS5506A for more information.

2.2 Parameterized Component Classifiers

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

A `Prototype_Substitution_Rule` property specifies matching rules for the classifiers that users can supply as prototype actuals. By default this rule matches classifiers, but other rules include type extension and signature match. Classifier match means that the classifiers must be identical, or the actual classifier may be an implementation of the component type specified as part of the prototype declaration. Type extension means that the actual classifier can be an extension of the classifier specified in the prototype declaration. Signature match means that for a component type the set of declared features of the actual classifier must contain at least the features specified in the classifier of the prototype declaration.

2.3 Classifier Substitution in Refinements

In AADL V1, users completed classifier references for subcomponent and feature declarations in `refined to` statements of component type and implementation extensions. In other words, users added a component type or, if they had already specified a component type, they added a component implementation.

In AADL V2, `refined to` for subcomponents (AADL V2 Section 4.5) and features (AADL V2 Section 8) also supports substituting classifiers with extensions of those classifiers. For example, users can substitute a component type with another component type that they declare directly or indirectly by an `extends` of the original component type. A `Classifier_Refinement_Rule` property indicates whether the classifier allows completion or substitution. The default is classifier matching.

2.4 Component Arrays and Connection Patterns

Users can declare subcomponents to be arrays (AADL V2 Section 4.5). These arrays can be single or multidimensional. Users can declare arrays 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 will replicate between the different elements of the source array and those of the destination array.

Users specify connection patterns through a `Connection_Pattern` property that they associate with the connection declaration. Its values determine, for each dimension of the array, how a source element in an array connects to a destination element. Figure 1 shows the resulting connections for single-dimensional source (S) and destination (D) arrays of three elements. The predefined patterns are one-to-one (identity), next, cyclic next, previous, cyclic previous, and all-to-all.

Users can combine patterns by listing multiple pattern values in the `Connection_Pattern` property.

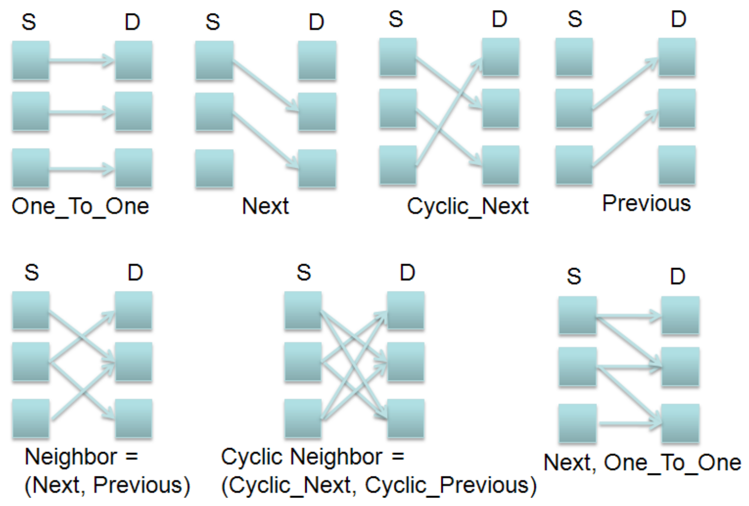

Figure 1: Connection Patterns for One-Dimensional Arrays

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.

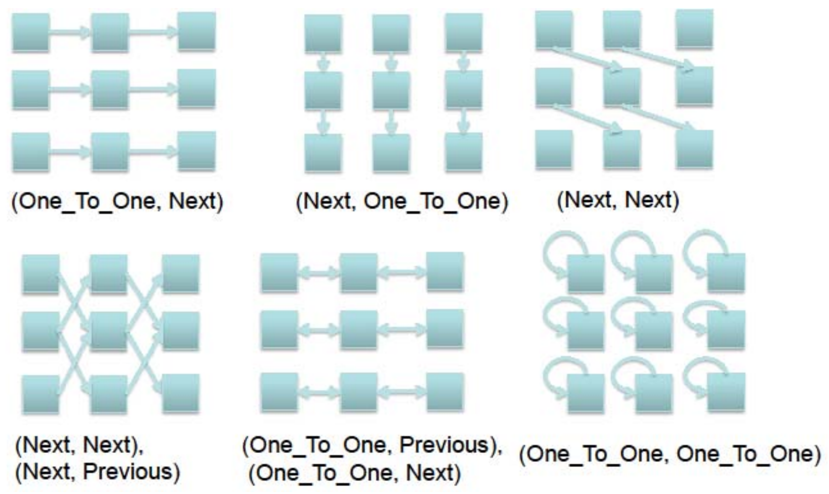

Figure 2: Internal Connection Pattern for a Two-Dimensional Array

If the predeclared pattern primitives and their combinations are not sufficient to express a desired connection topology, then through a `Connection_Set` property users can specify the topology of connections between the elements of both arrays as lists of array index pairs. Users may generate those values from a tool or by interpreting an algorithmic specification of the topology.

2.5 Improvements to Subprograms

In AADL V1, users could not declare subprogram subcomponents to model instances of code. This is now possible in AADL V2 (Section 5.2). It is optional—AADL V2 allows users to model systems with subprogram instances, but does not require it.

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, in which users document all interface requirements in the interface. As in AADL V1, users can declare calls and identify the subprogram that a binding property will call. 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. In addition, calls now refer to provided subprogram access in processors.

Users must name their call sequences. 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 threads (i.e., a thread that responds to both clock-based dispatches and event-based or 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. Improvements to the API specification of these service calls include an explicit parameter specification (AADL V2 Section 5.4.8).

2.8 Asynchronous Systems

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

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

Users can assign different reference times to processors, devices, buses, and memory through the `Reference_Time` property. The reference time for thread execution is determined by the reference time of the processor on which the thread executes. The reference time of communication among threads, devices, and processors is determined by the reference times of the source, destination, and, if it is time driven, any execution-platform component involved in the communication. An application may go to a time server to retrieve time for time-stamping data. Users express this by associating a reference time directly to the application component or by explicitly modeling the time server as part of the application.

2.9 Layered Architecture Modeling

AADL V2 contains a new section 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, system implementations represent the realizations of these abstractions and associate them with the component type or implementation declaration of processors, virtual processors, buses, virtual buses, memory, or devices. We added an `Implemented_As` property to specify the association of a system-implementation classifier with execution-platform classifiers. For example, a user might model the realization of a device such as a digital camera as a system implementation that consists of software, processors, memory, and charge-coupled sensor devices and that has the same interface as the declaration of the device type for the digital camera.

3 Feature and Connection Improvements

3.1 Abstract Features

AADL V2 introduces abstract features (AADL V2 Section 8.1) that represent placeholders for concrete features (i.e., ports, parameters, and the different kinds of access features). Users typically employ abstract features 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 uses feature prototypes (AADL V2 Section 4.7) to specify the concrete feature. These feature prototypes can be passed down the containment hierarchy.

Users can connect abstract features with feature connections (AADL V2 Section 9.1). Feature connections can also connect abstract features to concrete features.

3.2 No More `Refines Type`

In AADL V1, users could specify an implementation-specific property value for features by declaring feature refinements in the `refines type` subclause of a component implementation. AADL V1 restricted these feature refinements to the addition of property associations with features.

Users can express the same feature refinements through property associations contained in the `properties` subclause of the component implementation and identify the feature in the `applies to` clause of the property association. Therefore, we have removed the `refines type` subclause from AADL V2. The following example demonstrates:

```
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;
```

In this example we assume that the user defined the property `Integer_Range` in a property set called `Data_Model`. We standardized the declaration of such data-modeling properties in a Data Modeling Annex document for AADL (see Section 8.1).

3.3 Feature Groups

We revised the `port group` concept of AADL V1 to allow users to group any features and changed the keyword 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 users specify an `in` or `out` direction 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 the inverse of the port-group type separately. Port groups that will connect 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.

AADL V2 also supports declaration of a feature group that indicates it is the `inverse of` the feature-group type it references. This means the user does 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

AADL V1 limits port queues to in-event and in-event data ports of threads and devices. In AADL V2, users will continue to declare port-queue characteristics through properties in AADL V2. But now users can associate port queues with ports of enclosing components (thread groups, processes, and systems). This allows users to specify a port queue with a thread group or process that is serviced by multiple threads.

AADL V2 also allows users to associate port queues with out ports (AADL V2 Section 8.3.3).

3.6 Classifier Matching for Connections

AADL V1 requires users to make the types of the connection source and connection destination identical. AADL V2 allows the modeler to specify more flexible matching rules: classifier match (the default and the same semantics as AADL V1), equivalence (matching independently defined types), subset (the destination is considered to be a subset of the source type, e.g., to capture the specifications of the Object Management Group [OMG] Data-Distribution Service), 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 shorthand for declaring a collection of subcomponents. Users might need to connect this collection of components to a component that acts as a voter or an arbitrator of their output. They can make this connection by declaring the incoming feature as a feature array. AADL V2 limits this feature array to a single dimension, which users can declare for any type of feature.

When users connect a component array to a component with a feature array (e.g., an array of components 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 V2 (Section 9.2) now allows connections from data ports to event data ports and vice versa; between data or event data ports and data components, either directly or via data-access features; or from event data ports to event ports.

Connections between data ports of periodic threads now can be one of three kinds: immediate (mid-frame), delayed (phase delayed), and sampled (potentially nondeterministic sampling). Sampling is not available between periodic data ports in AADL V1; one has to use event data ports with a queue size of 1 to get that effect.

AADL V1 used the symbol -> for immediate connections and the symbol ->> for delayed connections. AADL V2 uses the `Timing` property instead of different connection symbols. Therefore, port connections can use only 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`.

Users can also now specify the port connection between an application component and a processor port.

3.9 Improved Communication Timing Specification

We added properties to allow 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 the dispatch time, then it is in terms of execution time.

Users can also specify input- and output-time properties for data-access connections. In that case, they specify the time ranges in which read access and write access occur to the shared data.

Rate properties allow users to specify input and output rates for individual ports. These rates 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 explicitly to initiate sending and receiving data through ports and the processing of port queues (AADL V2 Section 8.3.5).

3.11 Bidirectional Connections

AADL V2 allows bidirectional connections. The symbol -> represents unidirectional connections, and the symbol <-> represents bidirectional connections. Users can declare feature connections, port connections, access connections, and feature-group connections 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, we specified those connections 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 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 can also refer to events whose source is the component itself (**self**.eventname).

In addition, AADL V2 allows the arrival event of events, data, and event data to trigger mode transitions (i.e., in addition to event ports, users can name data ports and event data ports as triggers in mode transitions). This enhancement is consistent with the ability in AADL V2 to connect a data port with an event data port (see 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 users declare modes in a component type, they cannot add mode declarations to component implementations.

Mode declarations in the type also indicate whether modes are 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 can control mode switching by sending an event to the component. Declaring mode transition 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. Users will specify the inherited modes for a component type through a `requires modes` declaration (AADL V2 Section 12). The `in modes` clause of a subcomponent declaration specifies the mapping from the actual modes of the parent component to the inherited modes of the child component.

4.4 System-Level Mode Transitions

We improved the specification of a transition between two system-operation modes (AADL V2 Section 13.6). AADL V2 now supports 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 modes
2. get deactivated
3. get activated
4. are *zombie* threads (i.e., their execution has not completed yet at the time of the transition)

5 Packages and Visibility of Classifiers

5.1 With and Renames

The with clause specifies the set of packages that are acceptable qualifiers when users have specified classifiers (AADL V2 Section 4.2). In that situation, it limits the set of packages that users can name to those listed in the with clause (e.g., when declaring subcomponents). With clauses are declared for package sections and specify which property sets users can draw on for 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 be used only within the scope of the package in which it is declared
- must be unique within the namespace in which it is declared
- can be used instead of the qualified classifier references

5.2 Visibility of Component Implementations

In AADL V2, users can declare a component implementation in both the public and private parts of a package. The public part of a package reveals the identity of a component implementation, allowing users to name it in a subcomponent, while the private part of a package hides the details of the implementation. When this occurs, the component implementation in the public part contains only property associations and, if appropriate, mode declarations (AADL V2 Section 4.2).

In the following example, we declared the system implementations Gps.Dual and Gps.Secure 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 contain only 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 have properties. This enhancement provides finer control and makes the property mechanism accessible to sublanguage annexes. For example, a user can specify that a property applies to a `system type` (type), `system implementation` (implementation), `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.

AADL V2 also allows the user to define properties that apply to entities in an annex subclause. In other words, the property mechanism of the AADL core language is now available in annexes. For example, the Error Model Annex can define the `Occurrence` property in a property set and restrict its applicability to error events. Users specify applicability to entities in an annex subclause by naming classes in the annex Meta model (AADL V2 Section 11.1.2).

6.2 Revisions to Contained Property Associations

Contained property associations (AADL V2 Section 11.3) now

- allow a property value to be associated with multiple model elements (users can identify multiple model elements 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 component implementations, and with subcomponents. This allows users to associate properties with features and flow specifications that they declared in a component type or in a component type that is being extended.

6.3 Property Type Improvements

AADL V2 now supports record structures for properties (AADL V2 Section 11.1.1): 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.

Users should consider the set of enumeration literals in the definition of an enumeration type as an ordered list.

6.4 Property Values

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

Users can declare references as property values that 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 users do not predeclare, they will qualify the property name 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, AADL V2 interprets an identifier (following one of those keywords) as a literal. If users define an `enumeration` or `units` literal with the same identifier as a predeclared property and want to refer to the property instead of the literal, they can qualify the property 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`. AADL V2 can limit the applicability of properties to access features if users specify 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

AADL V2 contains several additional 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, users can specify `thread` entry points by naming a
  - subprogram in the source code
  - subprogram classifier
  - subprogram call sequence
- In AADL V2, users can reference all named elements of AADL models, including elements in annexes.
- We organized the predeclared properties 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 Namespace

The anonymous namespace in AADL V1 effectively provides a local workspace by allowing users to declare classifiers outside a package. AADL V1 considered the provision of workspaces the responsibility of the tool environment; in AADL V2, we eliminated the anonymous namespace. As a result, users must place all component types, component implementations, and feature-group types in packages.

7.2 Flows Through Shared Data Components

In AADL V1, users can specify flows for ports. AADL V2 extends flow specifications 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

Users can specify an end-to-end flow as a composition of other end-to-end flows, where the last element of the predecessor end-to-end flow connects 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 2006].

The SAE AADL Committee is in the process of revising and adding annexes, with three of them published in 2011 [SAE 2011]. In this section, we point to the updated and new versions of these annexes by their document letters.

8.1 Data Modeling Annex Standard

The Data Modeling Annex standard includes a standard set of properties and a collection of pre-declared basic data component types (AADL V2 Annex Document B) [SAE 2011].

8.2 Behavior Annex Standard

The Behavior Annex Standard allows modelers to annotate component types and implementations with behavior specifications (AADL V2 Annex Document D) [SAE 2011].

8.3 ARINC653 Annex Standard

The ARINC653 Annex standard provides guidance and sets properties that support modeling of partitioned architectures according to the ARINC653 standard [SAE 2011].

8.4 AADL Meta Model & XML Interchange Format Standard

The revised AADL Meta model & XML interchange-format standard provides a standard way of manipulating and interchanging AADL models. The OMG MARTE UML[4] profile for AADL is also based on the AADL Meta model (AADL V2 Appendix Document E).

8.5 Code Generation Annex Standard

We are developing a Code Generation Annex standard that provides guidance and a standardized set of properties to support automatic generation and integration of runtime systems and application components (AADL V2 Annex Document A).

8.6 Error Model Annex Standard

We are revising the Error Model Annex to support AADL V2, with publication expected in late 2011. It allows modelers to annotate component types and implementations with fault-behavior specifications, including probabilistic fault occurrence and propagation (AADL V2 Annex Document C). The original Error Model Annex standard was published in June 2006 [SAE 2006].

[4] MARTE stands for Modeling and Analysis of Real-Time and Embedded systems; UML is the Unified Modeling Language.

8.7 UML Profile for AADL via OMG MARTE

OMG MARTE has defined a UML profile for modeling embedded systems. The OMG MARTE document includes specifications for the AADL subset of MARTE as a standardized UML profile for AADL. The SAE AADL Committee will also approve this profile (AADL V2 Appendix Document F).

8.8 Future Annexes

We are considering several new annexes, which we have proposed to the SAE AADL Committee. They include a Requirements Definition & Analysis Annex, a Constraint Annex, and annexes for specialized architectures such as synchronous system architectures and time-triggered architectures.

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 Namespaces

In AADL V2, users must place all classifier declarations in packages. Thus, any component type, component implementation, port-group type, and annex-library declaration that users did not place in a package in AADL V1 must now 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 be referenced only within that package, there is no need for additional corrections.

AADL_specification ::=
{ AADL_global_declaration ~~| AADL_declaration~~ }$^+$

9.2 Package Declarations and Properties

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

Translation Action

Move the property associations into the single properties section:

package_spec ::=
package *defining*_package_name
(**public** package_declaration [**private** package_declaration]
| **private** package_declaration)
[**properties** ({ property_association }$^+$ | none_statement)]
end *defining*_package_name ;

package_declaration ::= { aadl_declaration }$^+$
~~[**properties** ({ property_association }+ none_statement)]~~

9.3 Package Declarations and `With` Clauses

In AADL V2, `with` clauses restrict the packages that users can reference by a given package.

Translation Action

Insert a `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 Subprogram Call Sequences

In AADL V1, call-sequence naming is optional, while AADL V2 requires it.

Translation Action

Add a generated name as the 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 because mode transitions do not have names.

In AADL V2, mode transitions can optionally have names, and users express a reference to a mode transition 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.

V1	Port group	Server subprogram
V2	Feature group	Provides subprogram access

9.8 Changes for Connections

Translation Action

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

V1	Data port	Event port	Event data port	Port group
V2	Port	Port	Port	Feature group

Also, for data-port connections, we no longer use the connection symbol 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 `with` clause restricts references to properties in property sets other than the predeclared properties to those listed in the `with` clause. AADL V1 has no such restriction.

Translation Action

Insert a `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 syntaxes using reserved words.

In AADL V2, these become identifiers in the AADL Meta model.

Translate V1	Into V2
For referable element categories	
Connections	Connection
Server subprogram	Subprogram access
For property owner categories	
Port group	Feature group, feature group type
Server subprogram	Subprogram access
Port group connections	Feature group connection
Event port connections	Port connection
Data port connections	Port connection
Event data port connections	Port connection
Port connections	Port connection
Access connections	Access connection
Parameter connections	Parameter connection

9.11 Changes in Property Expressions

AADL V1 requires an `access` reserved word for some properties. AADL V2 does not require this reserved word.

Translation Action

Remove this reserved word in property definitions and property associations.

In AADL V1, the value of another property used as the 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 the category name with `classifier`.

In AADL V1, a classifier term and a reference term do not require parentheses around the value. In AADL V2, we use `classifier`**(**<classifier name>**)**; for classifier terms and `reference`**(**<model element path>**)**; for reference terms.

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, we replaced them with `Access_Right`.

AADL V1 has no predeclared property called `Priority`. Instead, we defined this property 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 Conclusion

The changes that we 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 because we added virtual bus, virtual processor, and abstract components as component categories; component declarations now have explicit parameterization to better support templates and architecture patterns; V2 now supports component and feature arrays; package visibility is explicitly declared through `with` clauses; and properties can have record-structured values. We have also added a number of predeclared properties.

There is a simple mapping of existing AADL models into AADL V2, whose conversion is supported by an export capability in the OSATE toolset, Version 1.5.8.

References

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

[AADL Wiki 2011]
AADL Public Wiki with recent news, publications on use of AADL by various projects, and tools. https://wiki.sei.cmu.edu/aadl (April 2011).

[SAE 2004]
Society of Automotive Engineers. *SAE Standards: AS5506, Architecture Analysis and Design Language (AADL)*. http://www.sae.org/technical/standards/AS5506 (November 2004).

[SAE 2006]
SAE International. *SAE Standards: AS5506/1, Architecture Analysis and Design Language (AADL) Annex Volume 1*. http://www.sae.org/technical/standards/AS5506/1 (June 2006).

[SAE 2009]
SAE International. *SAE Standards: AS5506A, Architecture Analysis and Design Language (AADL)*. http://www.sae.org/technical/standards/AS5506A (January 2009).

[SAE 2011]
SAE International. *SAE Standards: AS5506/2, Architecture Analysis and Design Language (AADL) Annex Volume 1*. http://www.sae.org/technical/standards/AS5506/2 (January 2011).

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 2012	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE What's New in V2 of the Architecture Analysis & Design Language Standard?		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Peter H. Feiler, Joseph R. Seibel, and Lutz Wrage		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2011-SR-011
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

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). SAE International published the revised language, known as AADL V2, in January 2009. Feedback from users of the standard guided the plan for improvements. Their experience and suggestions resulted in the addition of component categories to better represent protocols as logical entities (virtual bus), scheduler hierarchies and logical time partitions (virtual processor), and a generic component (abstract). The revisions 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.

14. SUBJECT TERMS AADL, embedded systems, architectural modeling, real-time systems			15. NUMBER OF PAGES 34
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL