

# AADL Packages & Components

Peter Feiler

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Copyright 2018 Carnegie Mellon University. All Rights Reserved.

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

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

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.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material 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 use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM18-1285

# Packages for Property and Type Definitions

Request for property sets with nested identifiers

- Allow property definitions and type definitions in packages
- Decision: Yes

# Nested Packages

Package definitions have nested name paths

- Allow syntactic nesting of package declarations
- Qualified name of package is the combination of outer package names and defining package name

Decision: Yes

- Use <dot> as separator instead of ::
- Decision: Deferred

# Imported Namespaces

## Import declaration

- Make other package namespace content visible in a given package
  - All content: `Import packA.*;`
  - Specific definition: `import packB.TypeX [ as mine];`
- Declare within a package
- Reference by defining name only
  - Qualify if local definition with same name
  - Qualify if multiple imported definitions with same name

Decision: Yes

- Qualified name references are not required to be in listed in import declaration

Decision: Yes (Alexey,Jerome)

*Replaces **with** clause and **renames** declarations*

# Public and Private Sections in Packages

Public/private sections lead to complex rules about portions of implementation definitions residing in public and portions in private section

## Proposal

- Eliminate public and private sections in packages

## Proposal

- Allow classifier definitions to be marked as **private**

Decision: Yes

# Make AADL Case Sensitive

Identifiers: yes for all identifiers

Keywords:

- case insensitive (would not allow an identifier same as keyword other than case)
- Case sensitive – all upper and all lower; allows for identifiers with mixed case
- Case sensitive – ~~all upper or~~ **all lower**; allows for identifiers with mixed case (Yes)

Decision: Yes

# Section keywords in Classifiers

## Proposal

- Sections in arbitrary order: yes
- Eliminate sections with keywords
  - Revisit after nested components and **connection** keyword on connections

```
interface control is
insignal: in port;
outaction: out port;
processflow: flow path insignal -> outaction;
end;

process control.impl is
  dofilter: thread filter;
  docompute: thread compute;
  extin: map insignal -> dofilter.insignal;
  ftoc: port dofilter.outsignal -> docompute.insignal;
  extout: map outaction -> docompute.outsignal ;
  processflow => flow dofilter.filterpath -> ftoc -> docompute.computepath ;
end;

thread interface filter is
insignal : in port;
outsignal : out port;
filterpath: flow path insignal -> outsignal;
#Period => 20 ;
end;
```



# End keyword without Matching Name

## Proposal

- Eliminate matching name after **end** keyword
  - For packages
  - For classifier definitions

Recommendation: all but Brian

```
package PackC2
  type tt;
  interface mine is
    sig : in feature tt;
  end ;

  bus interface canbus end;

end ;
```

# Classifier Naming

As in AADL V2

Component interface name

- Single identifier

Component implementation name

- <component interface identifier> <dot> <impl identifier>

Configuration name

- <component interface identifier> <dot> <config identifier>

# Property Association

As before but with new syntax instead of *applies to*

```
[ ModelElementPath ] #<propertyname> => <property value>;
```

## General form used in classifier

Thread interface T is

  Inp: in port;

  #Period => 50 ms;

  Inp#Data\_Size => 6 Bytes;

End;

## In context of local declaration

Thread interface T is

  Inp: in port { #Data\_Size => 5 Bytes;;

End;

System s.impl is

  P1: process ComputeProcess.impl {

    #Code\_Size => 3.5 Kbytes;

    t1#Period => 20 ms;

    t2#Period => 10ms;

  };

End;

# Component Categories

## Category

- Once specified cannot be refined into another category
  - Binding better for mapping functions to implementation architecture
  - *May be useful for providing “implemented as”*
- Change **abstract** to **component** (Generic component)
  - *undecided*
- Usage: interface, implementation, subcomponent
- Category must match

## Component interface

- <category> and **interface** keyword
- Composable interface without category
  - Usage in interface composition
    - Content consistent with target category

```
interface sub
features
    name : in feature person ;
    surname : in feature person ;
end ;
process interface subsub
features
    p1 : port date ;
    p2 : port date ;
end ;
```

# Nested Subcomponent Declarations

## Nested components without explicit classifier

- Single instance of an unnamed classifier
- No interface enforcement at given level
- Reach down for connection declarations

Recommendation: proceed. Section labels inside nested declarations?

- implicit interfaces for intermediate nested component declarations

```
system ControlSystem {  
  sensing: device { sensedata: out port;};  
  processing: {  
    filter: thread {  
      inp: in port;  
      outp: out port;  
    };  
    control: thread {  
      inp: in port;  
      outp: out port;  
    };  
    filtercontrolconn: filter.outp -> control.inp;  
  };  
  actuating: device { inp: in port; };  
  sensefilterconn: sensing.sensedata -> processing.filter.inp;  
  controlactuateconn: processing.control.outp -> actuating.inp;  
}
```