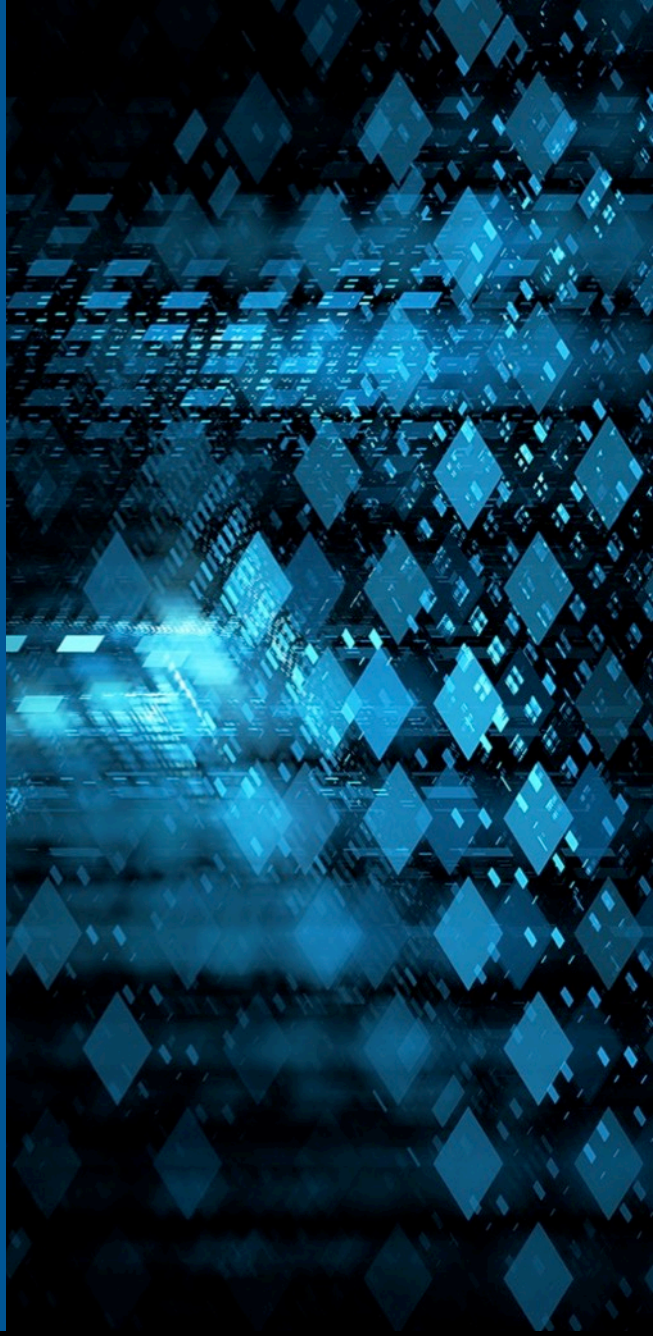
AADL Packages & Components

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



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

DM19-0166

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: Go with ::

Imported Namespaces

Import declaration

- Make other package namespace content visible in a given package
  - All content: Import packA::*; [alias for package name]
  - Specific definition: import packB::TypeX [**as** mine];
- Declare within a package
- Reference by defining name only
  - Qualify if local definition with same name (indicator to user)
  - Qualify if multiple imported definitions with same name
  - Alias can resolve multiple imported name conflicts

Decision: Yes including alias support

- 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

Recommendation: file per package (multiple nested packages ok). File name = package name.

Question: name nesting reflected in name nesting

Make AADL Case Sensitive

Identifiers: yes for all identifiers

Keywords:

- Case sensitive – ~~all upper xor~~ **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: mapping insignal => dofilter.insignal;
    ftoc: connection dofilter.outsignal -> docompute.insignal;
    extout: mapping 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"*
- 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. Think of this as pattern that needs to be satisfied by classifiers getting configured. Can we define implementations without an explicit type but identify path in nested structure. Name mapping of features

- Optional explicit interfaces for intermediate nested component declarations
  - Interface enforcement as design constraint?

```
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;
}
```

Optional semi-colon

Optional semi-colon for last in list of items

- List of properties in curly brackets (, vs ; as separator)
- List of nested subcomponents
- List of declarations in classifier (end as separator/terminator)
- Proceed

```
interface sub is
    name : in feature person;
    surname : in feature person ;
end ;
interface subsub is
    p1 : in port date ;
    p2 : in port date { #Data_Size => 3; };
    p1#Data_Size => 3;
end ;
```

```
tem 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;
}
```