

# AADL 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).

DM18-0659

# From Interface Composition

Component interface (aka type) and implementation

- For component type use **interface** keyword (stay with category only) when declaring types
- Optional component category

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

- For component implementation use **implements** keyword

- *System sub\_impl implements packA::sub*

```
system sub_impl implements sub
  subcomponents
    s1 : process subsub ;
    s2 : thread subsub { #Period => 20; };
  end ;
```

- Allow multiple interfaces in **extends**
- Implementation name is still type.impl as it provide syntactic feedback in subcomponents as to whether we have selected type or impl/config
- (PHF: some contexts allow implementation names & classifier.feature (subprogram))

# From Configurations

Allow reach-down configuration assignment (**refined to**)

- Substitution restricted to extensions in configurations
- No reach down into parameterized configurations
- Proposal to use  $\Rightarrow$  instead of **refined to** even in implementations
  - Substitution rule in configuration limited to configuration extensions
  - Category gets refined from abstract to a concrete one
- Bracketing {} vs. “end”, “section” keywords

Reach down property association without **applies to**

- **Sub1#period  $\Rightarrow$  20 ms;**

Configuration parameter replaces prototype

- Prototype requires reference of prototype throughout implementation
  - One level only requires repetition of prototype down the hierarchy
- Configuration parameter is mapped via reach down
  - Allows for parameterization of existing model without modification

**Extends** multiple to support composition of configurations

- Composition rules for configurations maintain/expand depth topology
  - Implementation extension substitution rule
    - Allow interface extension as well?
  - No additional components/connections are introduced
  - Flows, annexes, property values can be added

# AADL Components Revised

## Optional section keywords

- Currently: *features, properties, subcomponents, connections*
- Make optional or add into configuration
  - No ordering requirement for declarations (ok)
  - Classifier bracketing by {} or *begin / end* or *end only*

```
device interface sensor {  
    senseddata: out port;  
    #Period => 20;  
}
```

```
system gps {  
    location: out data port;  
    locator : process LocatorProcess ;  
}
```

# AADL Components Revised

## Implementation without separate interface declaration #12

- Allow features in implementation
  - For components with only one implementation
  - Extensions can add features

```
system gps
features
  location: out data port;
subcomponents
  locator : process LocatorProcess ;
end ;
```

- No: We still have typename.implname thus need to refer to type.

# AADL Components Revised

```

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

connections
    sensefilterconn : sensing.sensedata -> processing.filter.inp ;
    controlactuateconn : processing.control.outp -> actuating.inp ;
end ;
  
```

# AADL Components Revised

## Component category

- **Abstract** => **component**
- Eliminate **thread group** by allowing **component** or no category (no)
- Allow threads without being contained in **process**
  - Currently not allowed in **system** but in **abstract**
- Difference between **component** and **system**?
- For functional architectures
  - Use keyword **function**?



# Nested packages

Packages declared inside packages

- Grouping of packages
- Composite name path
- Scope of visibility of nested packages
- Import of package collection

# Optional subcomponent Category Keywords #13

## Subcomponent declaration without category

- Infer category from classifier
- Explicit category improves readability and allows for consistency check with classifier
- Explicit category allows association of category with generic classifier (interface or abstract/component)
- Category ensure additional semantics
- No (generic) category allows for component modeling without additional semantics (how about period property?)

# Composition of Implementations

We have composition of interfaces

We have composition of configurations