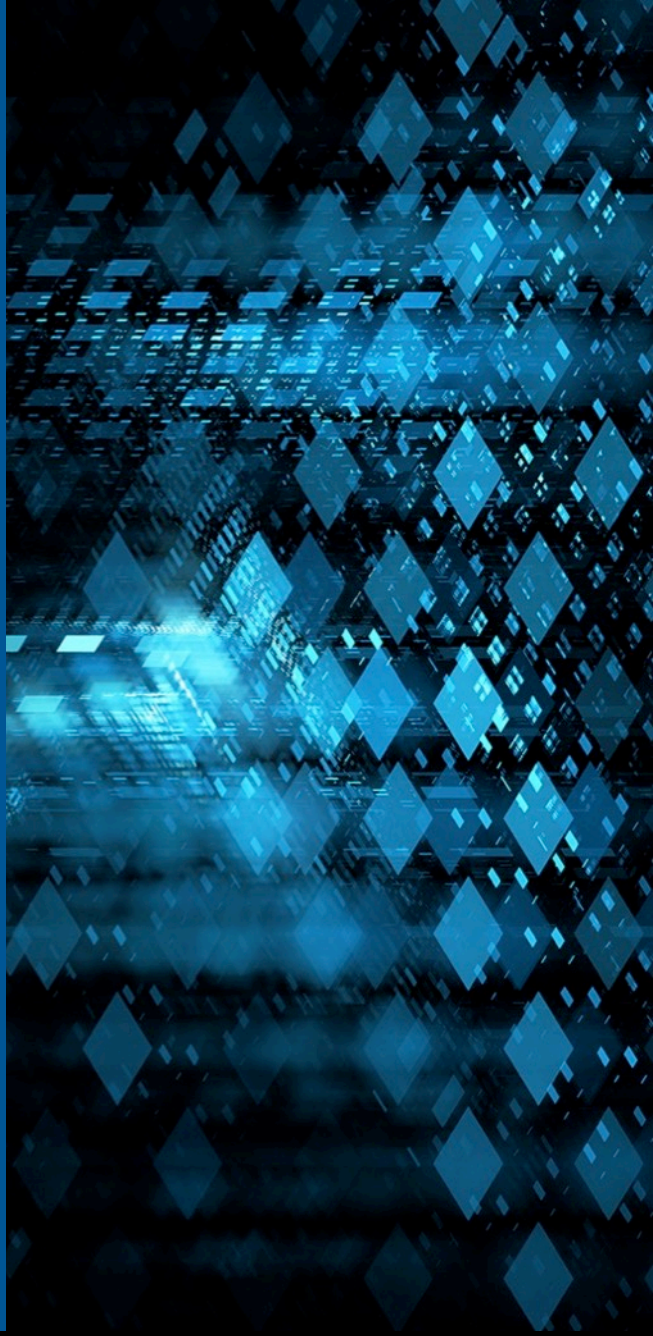
AADL Interface Composition

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

Composition of Interfaces

Objectives

- Definition of component interfaces by
  - Feature, flow, mode declarations and property associations
  - Extension of component interfaces through additional declarations in extension
  - Definition of component interfaces from previously defined composable interfaces
- Named interfaces as connection point

Approach

- Component interface declaration with *interface* keyword and optional component category
- Allow multiple component interfaces as part of extends
- Composition rules align with current extends rules
  - Local addition of elements in extension
- Named interface instances
  - Multiple instances of same interface replaces feature group concept in V2

Interfaces and Component Categories

Component interface

- <category> and **interface** keyword
  - *has implementations*
  - *referenced in subcomponent*
  - *Can be extended*
- **Interface** keyword without category (composable interface)
  - Usage in interface composition
    - Content must be 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 ;
```

Interface Extension

Extension and categories
- Defining interface and extended interface(s) must have same category or no category
- Extended interface can be an interface without category

Addition of features, flows, properties

Local refinement of inherited features in named interfaces
- Assign type when absent (primitive type or classifier)
- Override existing type with
  - Type extension
  - Any type

```
Interface Logical
 Temperature: out data port;
 AirPressure: out data port;
End Logical;

System interface mysys extends Logical
is
 Speed: out data port;
 Temperature => TemperatureData;
End;

System interface mysys1
is
  L1: Interface Logical{
    Temperature => TemperatureData;
  };
 Speed: out data port;
End;
```

Composition of Interfaces

Inherited content from multiple interfaces

- Cannot be in conflict (same as for local definitions)

```
interface Logical
is
temperature: out data port;
Speed: out data port;
End Logical;

interface Physical
is
Network: requires bus access CANBus;
End Physical;

interface s1 extends Logical
Onemore: out event port;
End s1;

interface s2 extends Logical, Physical
End s2;

interface s3 extends Logical, Physical
is
Onemore: out event port;
End s3;
```

Right: at most one with category and others composable

V2: Locally added feature cannot conflict with a feature inherited from Logical

V3: Feature from Logical and Physical cannot be in conflict

V3: Locally added feature cannot conflict with inherited features

Composition of Directional Interfaces

Interfaces with directional features may be included as original direction or as inverse direction for component at the other end of a connection

- This is the inverse of from feature groups

```
System interface Sender extends Logical, Physical
End;

System interface Receiver extends Physical, reverse Logical
End;
```

Composition of Named Interfaces

Objective: Handle multiple instance of same interface, e.g., voter taking input from multiple instances of same subsystem

- Individual features qualified by interface instance name
- Internally: interfaceinstancename . Featurename
- Externally: subcomponentname . interfaceinstancename . Featurename
- Connections between named interfaces

Directionality of arrow on named interface:
Bi-directional arrow for interface connection.
Connections between directional features must be directional.
Directional connection on bi-directional interface: no.

```
System interface sif1
    IFlog: interface Logical;
    IFphys: interface Physical;
End;
System interface voter
Source1: interface reverse Logical;
Source2: interface reverse Logical;
End;

System Top.impl is
Sub1: system sif1;
Sub2: system sif1;
Voter: system voter;

Conn1: connection Sub1.IFlog <-> Voter.Source1 ;
Conn2: connection Sub2.IFlog.temperature -> Voter.Source2.temperature ;
End;
```

Connections between named interfaces (V2 feature group connections) or between features in an interface (reach down V2.2)

Use of Named Interfaces

Example of mapping output to ports in different named interfaces

```
Device sensor is
temperature: out data port;
Speed: out data port;
End;

System sys2
is
  L1: interface Logical;
  L2: interface Logical;
Fl: flow L1.outp -> L2.inp;
End sys2;

System sys2.i1 is
  sub1: device sensor;
  conn1: sub1.temperature -> L1.temperature;
  conn2: sub1.temperature -> L2.temperature;
End;

System sys2.i2 is
 sub1: device sensor;
 sub2: device sensor;

  conn1: sub1.temperature -> L1.temperature;
  conn2: sub2.temperature -> L2.temperature;
End;
```

How to refer to flow inside Logical?
L1.p1#DataSize =>

sub1 output is mapped into a port in two different interfaces. These may be ports with the same name, or ports with different names.

Output from different sources to different interfaces. L1.temperature and L2.temperature receive different output.

Nested Interfaces

Works for composition of named interface instances

- Nested name scopes
- Effectively we have nested feature groups
- Deprecate feature groups in V3

```
Interface composite is
  L1: interface Logical1;
  PF: interface Physical;
End;

System interface Top is
  FG: interface composite;
  L2: interface Logical2;
End;
```

All features in single namespace

Unnamed interfaces share a name space (no nested name space)

```
Interface composite extends Logical1, Physical
End composite ;

System interface Top extends composite, Logical2
End top;
```

Name conflict between Logical1 and Logical2 feature temperature

Subcomponent Refers to Interface

Substitution of any component that is an extension of interface

- Only in implementation extensions (not in configurations)
- Allow multiple interfaces on right hand side (unnamed composite interface)

- Rules about connected port (port_connection property)

```
System interface Sensor extends Logical, Physical
End;
System interface Actuator extends reverse Logical, Physical
End;

System Actuator.impl
End;


System top.i is
  sub1: system Logical;
  sub2: system reverse Logical;
  conn1: sub1.temperature -> sub2.temperature;
End;

System top2.i extends top.i
is
  sub1 => Sensor;
  sub2 => Actuator.impl;
<connections to additional features>
End;
```

Assign a component classifier that supports the interface plus more

Composition of Interface Property values

Interface property values are inherited by the component

```
Thread Interface Logical is
temperature: out data port;
Speed: out data port;
#Period=> 10ms;
Speed#Rate => 5 mpd;
End;

Interface Physical is
Network: requires bus access CANBus;
#Period => 10ms; -- should this property be there?
End;

System s2 extends Logical, Physical
End;

System s3 extends Logical, Physical is
#Period=> 20ms;
End;
```

Component level property value

Feature level property value

One inherited assignment only: Yes

Multiple inherited assignments of same value: No

Subject to default, final, override rules

Composition of Interface Property Values - 2

Named interface composition

- Component level property values apply to component, not the named interface name space

```
Interface Logical is
temperature: out data port;
Speed: out data port;
#Myname => "peter";
End;

Interface Physical is
Network: requires bus access CANBus;
Properties
#Hisname => "peter";
End;

System s2 is
 L1: Interface Logical;
 P1: Interface Physical;
L1#DataSize => 30 Bytes;
End s2;
```

Component level property value

Myname and Hisname are s2 properties, not L1 and P1 properties.

Composition of Flows

Same rules as V2 extends
Flows in interfaces are only with respect to its features
The composite component may add flow specification for flows between features in different interfaces

```
Interface Logical
temperature: out data port;
Speed: out data port;
flows
 temp: flow source temperature;
End Logical;

System s2 implements Logical, Physical
flows
 spd: flow source speed;
End s2;
```

Can add flows for inherited features as was possible in V2

Composition of Modes

Only one source (same as **extends** of single classifier)

- Local additions as in V2
  - current std allows adding states in type extensions

Annex Composition

Configuration of annex specifications into an AADL model

- See configuration discussion

Composition of annexes from different interfaces

- Same Annex notation in two interfaces
  - Not allowed
- Local addition of annex
  - Follow annex rules for annex extension

Feature Name Mapping for Connections

Support for composition of independently developed subsystems or subsystem with different nested interface hierarchies

- Inline mappings (reach down multiple interface nesting levels)

```
Conn1: sub1.lfea1.fea2 -> sub2.rfea1;
Conn2: sub1.lfea1.fea3 -> sub2.rfea2.fea11;
Conn3: sub2.rfea2.fea12 -> sub1.lfea1.fea4;
```

Needs to be repeated for each pair of subcomponent instances

- Reusable equivalence mapping

```
map1: mapping ComponentType1 == ComponentType2 as
lfea1.fea2 == rfea1;
Lfea1.fea3 == rfea2.fea11
end mapping ;
```

Name mapping between name scope hierarchies

Direction is inferred from connection declaration and feature direction.

```
Connx: sub1 -> sub2 mapping pckx::map1;
```

Is reusable mapping needed? Alternative: use name mapping in a feature mapping (up/down) as a wrapper or in an enclosing component with mapping between them.

Use as Aggregate Port

Interface elements interpreted as elements of aggregate data

```
Device sensor is
temperature: out data port;
Speed: out data port;
End;

System sys2
is
  L1: aggregate Logical;
End sys2;

System sys2.i1 is
  sub1: device sensor;
  conn1: sub1.speed -> L1.speed;
  conn2: sub1.temperature -> L1.temperature;
End;
```

Use output rates etc on aggregate.

For implementation architecture use virtual bus as an aggregator. Its binding indicates over what part of the HW flow it stays aggregated.

Do we need aggregate port specifications?

Should this be a protocol issue?