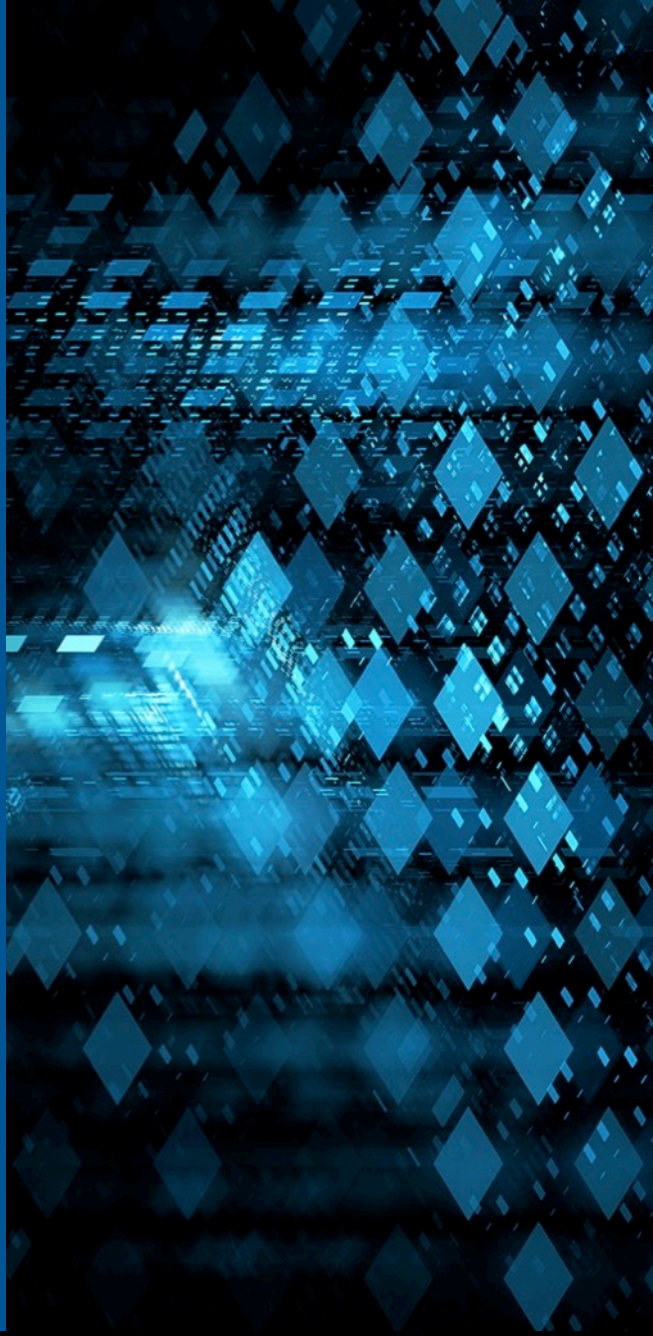
AADL Interface Composition

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.

DM18-0659

Composition of Interfaces

Objectives

- Composition of features and properties into a component type (interface)
- Named interfaces as single connection point
- Composition rules for features, modes, flows, annexes
- Declaration of composed interface or implementation supporting multiple interfaces

Approach

- Component interface (type) 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
- Interfaces as views
  - Elements with same name are ok (must represent same element)

Composition of Interfaces

Features accessible directly within namespace of component

- Externally: connections identify subcomponent and feature (V2)
- Internally: connections identify feature (V2)

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

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

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

interface s2 extends Logical, Physical
End s2;

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

V2: Locally added feature cannot conflict with a "Logical" feature

V3: Feature from Logical and Physical cannot be in conflict

V3: Locally added feature name cannot exist as name of Interfaces 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 sender;

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

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

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

System Top_impl
Subcomponents
Sub1: system sif1;
Sub2: system sif1;
Voter: system voter;
Connections
Conn1: Sub1.IFlog -> Voter.Source1 ;
Conn2: Sub2.IFlog.temperature -> Voter.Source2.temperature ;
```

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

Composition of Named Interfaces

Objective: Handle interfaces with independent features with same name

```
Device Logical1
temperature: out data port;
Speed: out data port;
End;

Abstract Logical2
temperature: out data port;
weight: out data port;
End;

System sys2
features
  L1: feature group Logical1;
  L2: feature group Logical2
End sys2;
System sys2_i1 implements sys2
Subcomponents
  sub1: system sub;
Connections
  conn1: sub1.outp -> L1.temperature;
  conn2: sub1.outp -> L2.temperature;
End;
System sys2_i2 implements sys2
Subcomponents
 sub1: system sub;
 sub2: system sub;
Connections
  conn1: sub1.outp -> L1.temperature;
  conn2: sub2.outp -> L2.temperature;
End;
```

In the implementation the connection declarations specify that the same sub1 output is mapped into a port in two different interfaces. These may be ports with the same name, or ports with different names.

L1.temperature and L2.temperature may receive output from two different internal sources.

Different output to different interfaces

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
features
  L1: interface Logical1;
  PF: interface Physical;
End composite ;

System Top
features
  FG: interface composite;
  L2: interface Logical2;
End top;
```

All features in single namespace

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

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

System 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 type

```
System mysys extends Logical, Physical
End mysys;

System mysys1
Features
 L1: interface Logical
End mysys;


System top_i
Subcomponents
  sub1: system Logical;
  sub2: system Logical;
Connections
  conn1: sub1.outp -> sub2.inp;
End top.i;

System top2_i extends top_i
Subcomponents
  sub1 => mysys;
  sub2 => mysys1.L1;
End top.i;
```

Configure in a component that implements the interface

Configure in a component with a named interface that matches. Connections to subcomponent actually go to named interface.

Composition of Interface Property values

Interface property values are inherited by the component

```
Interface Logical
features
temperature: out data port;
Speed: out data port;
Properties
#Author=> “peter”;
Speed#Rate => 5 mpd;
End Logical;

Interface Physical
Network: requires bus access CANBus;
Properties
#Author => “peter”;
End Physical;

System s2 implements Logical, Physical
End s2;

System s3 implements Logical, Physical
properties
#Author=> “paul”;
End s3;
```

Component level property value

Feature level property value

Same property from two interfaces must have same value

Can override property locally. Can be used to resolve conflict of inherited values.

Composition of Interface Property Values - 2

Named interface composition

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

```
Interface Logical
features
temperature: out data port;
Speed: out data port;
Properties
Myname => “peter”;
End Logical;

Interface Physical
Network: requires bus access CANBus;
Properties
Hisname => “peter”;
End Physical;

System s2
Features
 L1: Interface Logical;
 P1: Interface Physical;
End s2;
```

Component level property value

Myname and Hisname are s2 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

- Follow annex rules for annex extension
  - Addition and override in extension
- Same name in two interfaces
  - Not allowed
  - Same specification

Interfaces as Views

Features may show up in multiple interfaces

- Some features may be available in both an “admin” and an “operational” view

```
Interface Functional
features
temperature: out data port;
Speed: out data port;
Reset: in event port;
End;

Interface Admin
features
Status: out data port;
Reset: in event port;
Shutdown: in event port;
End;

System full extends Functional, Admin
End;
System overlap extends Functional, Functional
End;
```

V3: Matching features from Logical and Admin represent same feature

Matching name, category and classifier (if present)

Matching property values

Features differ in the classifier

- **no classifier to classifier => must provide classifier.**
- **Allow classifier extension => must provide extension.**

Views traditionally represent subsets of existing entities.

Here we are composing overlapping views.

Should view be a separate concept? Do we need it?

Are named interfaces sufficient?

Should users indicate that the overlap is intentional?

Feature Refinement & Named Interfaces

Local refinement of inherited features in named interfaces

```
Interface Logical
temperature: out data port;
Speed: out data port;
End Logical;

System mysys extends Logical
Features
 temperature: refined to out data port TemperatureData;
End;

System mysys1
Features
  L1: Interface Logical{
    temperature => TemperatureData;
  };
End;
```

Configuration syntax

Refinement of Composite Interface

Use of refined interface in composition

```
Interface Logical1 extends Logical
Features
temperature: refined to out data port TemperatureData;
End Logical1;

System mysys implements Logical, Physical
End mysys;

System mysys2a implements Logical1, Physical
-- no extends trace to mysys
End mysys2;
```

Composition with refined interface without traceability to mysys, thus implementation has to be repeated

```
System mysys2b extends mysys implements Logical1, Physical
End mysys2;
```

Interfaces override

```
System mysys2b extends mysys implements Logical => Logical1
End mysys2;
```

Interfaces inherited

```
System mysys1typeda extends mysys1
End mysys1typeda ;
System mysys1typedb extends mysys1
Features
l1: interface refined to Logical1;
End mysys1typedb ;
```

Refinement of named interface

Interface Equivalence Mapping

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

- Inline mappings (reach down multiple 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?