# 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.

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.

DM18-0076

# Composition of Interfaces

Objectives

- Composition of features and properties into a component type
- Single connection declaration for interfaces aka feature group
- Composition rules for features, modes, flows, annexes

Approach

- Allow extends of multiple component types
- Composition rules align with current extends rules
  - Composition of abstract category to become abstract or specific component category
  - Composition of specific component category into the same category
- Allow multiple named instances of the same interface
  - Effectively offers nested feature group connectivity

# Composition of Interfaces

Features accessible directly within namespace of component

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

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

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

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

System s2 extends Logical, Physical
End s2;

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

V2: Locally added feature name cannot exist as name of a “Logical” feature

V3: Feature ~~names~~ from Logical and Physical cannot be in conflict

V3: Locally added feature name cannot exist as name of Interfaces features

# Interfaces as Views

Features may show up in multiple interfaces

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

```
Abstract Logical
features
temperature: out data port;
Speed: out data port;
Reset: in event port;
End Logical;

Abstract Admin
features
Status: out data port;
Reset: in event port;
Shutdown: in event port;
Pp => dt;
End admin;

System s1 extends Logical, Admin
End s1;
```

**V3: Matching features from Logical and Admin represent same feature**

**Matching name, category and classifier (if present)**

**We allow same property value in different interfaces**

# 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 Sender extends Logical, Physical
End sender;

System 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 sif1 extends
    IFlog: Logical,
    IFphys: Physical
End sif1;
System voter Extends
Source1: reverse Logical,
Source2: reverse Logical
End voter ;

System implementation 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

## Alternative syntax for named interfaces

- Named interface in same name space as other feature declarations
- Same as we had for feature group declaration

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

System implementation 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 ;
```

**Need for keyword? Candidate:** interface

# Composition of Named Interfaces

Objective: Handle interfaces with independent features with same name

```
Abstract Logical1
temperature: out data port;
Speed: out data port;
End Logical;

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

System s2 extends L1: Logical1, L2: Logical2
End s2;
System implementation s2.i
Subcomponents
  sub1: system s1;
Connections
  conn1: sub1.out -> L1.temperature;
  conn2: sub1.out -> L2.temperature;
End s2.i;
System s3 extends Logical1, Logical2
End s2;
System implementation s3.i
Subcomponents
  sub1: system s1;
Connections
  conn1: sub1.out -> temperature;
End s3.i;
```

**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.**

**Use : as separator as we reach into elements of named interface?**

**Same output available in both interfaces (views).**

# Feature Refinement & Named Interfaces

## Local refinement of inherited features in named interfaces

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

System mysys extends Logical
Features
  temperature: refined to out data port TemperatureData;
End mysys;
System mysys1 extends L1: Logical
Features
  L1.temperature: refined to out data port TemperatureData;
End mysys;
```

# Refinement of Composite Interface

## Use of refined interface in composition

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

System mysys extends Logical, Physical
End mysys;

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

**Composition with refined interface from scratch without tracability to mysys**

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

**Logical in mysys and Logical1 are merged according to view rules.**
**Match rule: no classifier to classifier => must provide classifier.**
**Allow classifier extension => must provide extension.**

```
System mysys1typeda extends mysys1, l1 => Logical1
End mysys1typeda ;
System mysys1typedb extends mysys1
Features
l1: refined to Logical1;
End mysys1typedb ;
```

**Refinement of named interface. This syntax goes along with declaring named interfaces in the features section**

# Subcomponent Refers to Interface

Substitution of any component that is an extension of interface type

- Type_Extension substitution rule

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

System mysys1 extends L1: Logical
End mysys;


System implementation top.i
Subcomponents
  sub1: system Logical;
  sub2: system Logical;
End top.i;

System implementation top2.i extends top.i
Subcomponents
  sub1: refined to system mysys.i;
  sub2: refined to system mysys1.L1;
End top.i;
```

# Nested Interfaces

Works for composition of named interface instances

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

```
Abstract composite extends L1: Logical1, PF: Physical
End composite ;

System Top extends FG: composite, L2: Logical2
End top;
```

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

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

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

**All features in single namespace**

**Name conflict between Logical1 and Logical2 feature temperature**

# 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;
```

**Directional connection syntax**

**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;
```

# Composition of Properties

Specify a set of property values

- That apply to features in the interface or contribute to the component that supports the interface
- Properties section of interface
  - Equivalent of data set (mini) annex

Specification of which properties apply to a component

- Stereo type identifies a set of property definitions
  - May or may not include a property value
  - Gets associated with component classifier (or other model element)
- Component can have multiple associated stereo types
  - Property definition reference in multiple stereo types is acceptable

# Composition of Interface Property values

Interface property values are inherited by the component

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

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

System s2 extends Logical, Physical
End s2;

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

**Component level property value**

**Feature level property value**

**Same syntax as for configuration**

**Same property from two interfaces must have same value (ok last time)**

**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

```
Abtract Logical
features
temperature: out data port;
Speed: out data port;
Properties
Myname => “peter”;
Rate => 5 mpd applies to Speed;
End Logical;

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

System s2 extends L1: Logical, P1: Physical
End s2;
```

Component level property value

Feature 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

Qualified name for flows when extends reference is qualified

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

System s2 extends Logical, Physical
End s2;

System s3 extends l1: Logical, Physical
flows
 spd: flow source speed;
End s3;
```

**Use of named extends references addresses possible name conflicts.**

**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