Composition of Interfaces

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Peter Feiler

July 2016

Composition of Interfaces

Objectives

- Composition of interfaces and properties into a component type
- Single connection declaration for interfaces
- 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
temperature: out data port;
Speed: out data port;
End Logical;

Abstract Physical
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 be in conflict with Logical features

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

V3: Features from Logical, Physical and local cannot be in conflict.

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 s1;

System Receiver extends inverse of Logical, Physical
End s2;
```

inverse of for Logical to get opposite port direction
No inverse of for Physical since both require access to physical platform
Use another term instead: conjugate (Hecht)

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
  - Exernally: subcomponentname . interfaceinstancename . Featurename

```
System sif1 extends
    IFlog: Logical,
    IFphys: Physical
End sif1;
System voter Extends
Source1: inverse of Logical,
Source2: inverse of Logical
End s2;

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

Objective: Handle interfaces with conflicting feature names

```
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 2s.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.

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 mysys1 extends mysys
Features
temperature: refined to out data port TemperatureData;
End mysys1;

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

System mysys3 extends mysys (Logical refined to Logical1)

End mysys3;
```

Nested Interfaces

Works for composition of named interface instances

- 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

```
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;
```

- Reusable equivalence mapping

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

Composition of Interface Properties

Interface properties are inherited by the component

- Also for nested interfaces the properties
- In feature groups the feature group type property did not become a component property but an inheritable property of the feature group

```
Abtract Logical
temperature: out data port;
Speed: out data port;
Properties
Myname => “peter”;
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
Myname => “paul”;
```

Cannot inherit same property from two interfaces: equal is ok

Can override property locally

Feature Group Types and Feature Groups

Feature group types define sets of features that can be instantiated as feature groups.

Properties can be associated with features in feature group type.

Property associated with feature group type is inherited by all elements in group.

Feature group declarations define named instances of feature collections that can be referenced in connection declarations as a connection of the collective.

Proposal: Eliminate feature group types. Named interface instance declarations replace feature group declarations.

Property lookup for inherit goes to the enclosing component (skip lookup in enclosing feature group type). The properties defined in an interface are associated with the enclosing component.

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

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

Cannot inherit flows with the same name. Use of named interfaces 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

Union of mode states from different interfaces

- Were they developed independently for the same state machine or as independent state machines?

Separate mode state machines

- We currently do not support multiple concurrent state machines for the same component
- Note that modal subcomponents have the effect of concurrent state machines