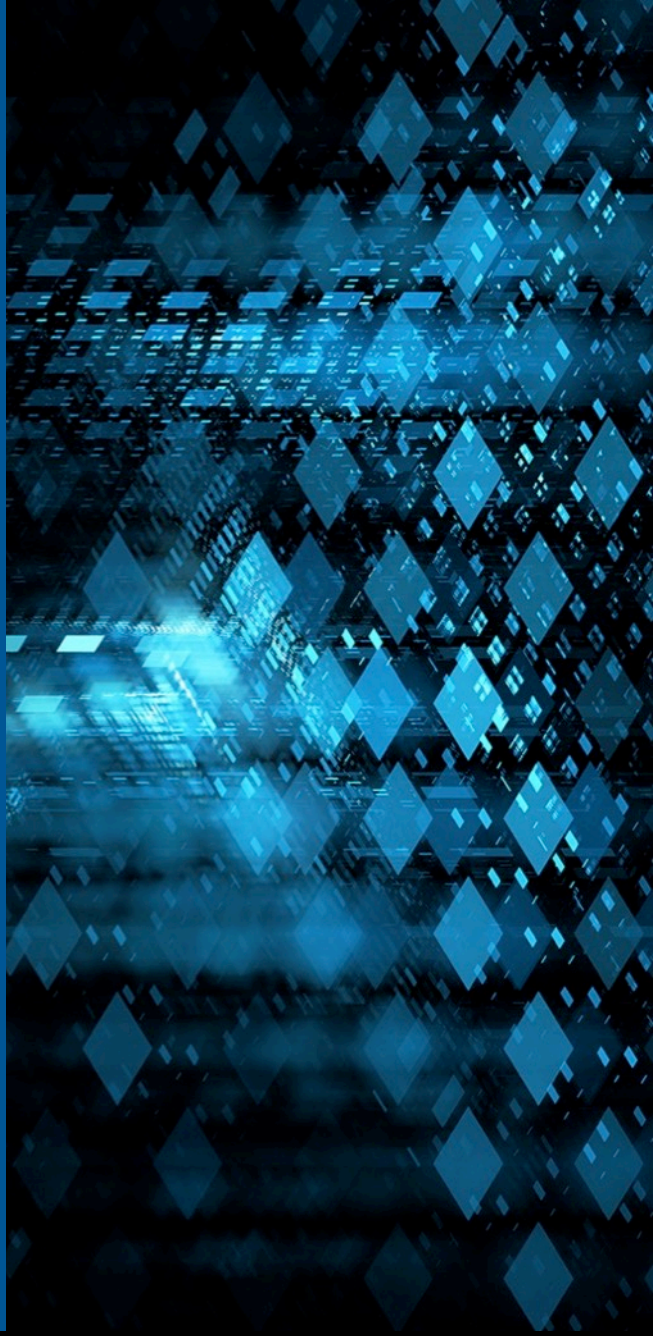
AADL Configuration Specification

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

Architecture Design & Configuration

Architecture design via extends, refines to evolve design space (V2)

- Revise and add to existing architecture design structure
- Add/revise annotation of property values, bindings, annexes

Configuration specification

- Elaborate but do not change architecture structure
- Configuration assignments
  - implementation to subcomponents
  - Types or classifier to features
  - Association of collections of final property values, bindings, annexes to given architecture substructure

Composition of configuration specifications

Parameterized configuration specification

- Subcomponent configuration assignment via parameter only

Evolution of System Design

Component Interface Extension

- Addition of features, flows, etc.
- Assignment of types/classifiers to existing features
  - Assign missing type
  - Override with type extension or any type
    Decision:
- Assignment of property values

Myport => MyDataType;
Same as configuration assignment syntax

Extension without feature addition:
Difference to interface configuration?

Component Implementation Extension

- Addition of subcomponents, connections, etc.
- Revision of existing subcomponents
  - Assign implementation for specified interface
  - Override existing implementation with extension
  - Override existing implementation with alternative
  - Assign interface extensions and their implementations

V2 type match allows implementation override

V2 type extension

Eliminate signature match and need for substitution rule specification
Decision:

Configuration of a System Design

Configuration Specification elaborates and annotates component hierarchy

- Associated with an implementation/interface via **extends**
- Configuration assignment assigns
  - implementation or configuration to subcomponent
  - Data type or classifier to feature
- Assign “final” property values within existing component hierarchy
- Specify bindings
- Add flow specification
- Add annex subclauses

Configuration Assignment

Configuration assignment

- Elaborate and annotate subcomponent substructure
  - Annotate substructure with “final” property values, bindings, annex subclauses
  - Assign component implementation for subcomponent with interface
    - Explicit: it becomes the intended implementation
    - Via configurations: associated implementation
    - Cannot be for interface extension

```
configuration Top.config_L1 extends top.basic
is
Sub1 => x.i;
Sub2 => y.i;
end;
```

```
System top.basic is
 Sub1: system x;
 Sub2: system y;
End;
```

Replacement of interface by implementation or configuration

Configuration of a System Design

- Assign configurations for subcomponent with implementation
  - Configurations for ancestor implementation or interface are ok

```
configuration Top.config_L1 extends top.L1impl
is
Sub1 => x.i2;
Sub2 => y.performance;
end;
```

```
System top.L1impl is
 Sub1: system x.i;
 Sub2: system y;
```

```
System x.i is
 xsub1: process subsubsys;
 xsub2: process subsubsys;
```

```
System x.i2 extends x.i is
 xsub3: process subsubsys;
```

```
System y.i is
 ysub1: process subsubsys;
 ysub2: process subsubsys;
```

```
configuration y.performance extends y.i is
 xsub1#Period => 20 ms;
```

Should we allow implementation extension as part of configuration assignment in a configuration specification? It potentially adds additional subcomponents

Configuration Across Multiple Levels

- Reach down configuration assignments
  - Left hand side resolved relative to classifier being extended

```
configuration Top.config_Sub11 extends top.L1impl
is
  Sub1.xsub1 => subsubsys.i;
  Sub1.xsub2 => subsubsys.i;
end;
```

```
System top.L1impl is
 Sub1: system x.i;
 Sub2: system y.i;
```

```
System x.i is
 xsub1: process subsubsys;
 xsub2: process subsubsys;
```

Nested Configuration Assignment

- Nested configuration specification
  - Used to configure an assigned classifier
  - Left hand side resolved relative to enclosing extended or assigned classifier

```
System x.l2 extends x.i is
xsub1 => subsubsys.i;
 xsub2 => subsubsys.i;
```

```
configuration Top.config_Sub1 extends top.basic
is
  Sub1 => x.i {
    xsub1 => subsubsys.i;
    xsub2 => subsubsys.i;
  }
end;
Sub1 => x.l2
```

```
System top.basic is
 Sub1: system x;
 Sub2: system y;
```

```
System x.i is
 xsub1: process subsubsys;
 xsub2: process subsubsys;
```

  - Nested configuration for existing subcomponent classifier

```
configuration Top.config_Sub11 extends top.L1impl
Is
Sub2 => {
    ysub1 => subsubsys.i;
    ysub2 => subsubsys.i;
    annex EMV2 {** … **};
    #Period => 20 ms
  };
end;
```

Shorter target paths

Annex assignment without explicit configuration specification

Property assignment without target path

Assignment of Configuration Specifications

Specification and use of separate subsystem configurations

- Configuration of subsystems

```
Configuration x.config_L1 extends x.i is
  xsub1 => subsubsys.i;
  xsub2 => subsubsys.i;
end;
Configuration y.config_L1 extends y.i is
  ysub1 => subsubsys.i;
  ysub2 => subsubsys2.i;
end;
```

- Use of configuration as assignment value

```
Configuration Top.config_L2 extends top.basic is
  Sub1 => x.config_L1;
  Sub2 => y.config_L1;
end;
```

Implementation associated with configuration is assigned to the target subcomponent if the original assignment is an interface

```
Configuration Top.config_L1L2 extends top.L1impl is
  Sub1 => x.config_L1;
  Sub2 => y.config_L1;
end;
```

Implementation associated with configuration must be the same **or an ancestor** of the original implementation

Configuration of Property Values

Specifying a set of property values

- Property value assignment to any component in the
  - subcomponent path resolvable via the classifier referenced by **extends**
  - Assigned value is “final”
  - May override previously assigned “default” values

```
Configuration Top.config_Security extends Top.config_L2
is
  #myps::Security_Level => L1,
  Sub1#myps::Security_Level => L2,
  Sub1.xsub1#myps::Security_Level => L0,
  Sub2#myps::Security_Level => L1
end;

Configuration Top.config_Safety extends Top.config_L1
is
  #myps::Safety_Level => Critical,
  Sub1#myps::Safety_Level => NonCritical,
  Sub2#myps::Safety_Level => Critical
end;
Configuration x.config_Performance extends x.i
is
  xsub1 => subsubsys.i {
   #Period => 10ms,
   #Deadline => 10ms }
end;
```

A configuration specification may only annotate property values or it may also configure and annotate other items.

Composition of Configurations

Combine multiple configurations into new configuration specification

- Define configuration with multiple extends
- Multiple configuration assignments to same subcomponent

Rules

- Associated interfaces must be the same
- Associated implementations must have a single extends lineage
  - The implementation associated with the composite: most descendant
- Only one property value assignment is allowed for any assignment target
  - Property value assignments in configuration specifications are “final”

```
Configuration Top.config_L2 extends top.config_L1, Top.config_Sub1, Top.config_Sub2 end;

Configuration Top.config_L22 extends Top.config_Sub1, Top.config_Sub2 end;


Configuration Top.config_SafeSecure extends Top.config_L2, Top.config_Safety,
  Top.config_Security end;

Configuration Top.config_SafetySecurity extends Top.config_Security, Top.config_Safety end;
```

Unnamed Compositions

Unnamed composition as part of a subcomponent configuration

- Same rules as for composite configuration specification (Probably yes)

```
Configuration Top.config_L2 extends top.basic is
  Sub1 => x.config_L1;
  Sub1 => x.security;
  -- shorthand:  Sub1 => x.config_L1, x.security;
  Sub2 => y.config_L1;
end;
```

Multiple assignments to same target act as implicit composition.

Unnamed composition as part of a subcomponent declaration

- Same rules as for composite configuration specification (probably not)

```
system top.basic is
 Sub1: process proc.i , proc.safety;
 Sub2: process proc.security , proc.safety;
 end;
```

Implicit composition (unavoidable)

- Different assigned configurations may contain configuration assignment to same target component
- Same rules as for composite configuration specification

Composition of Flow Configurations

Adding in end to end flows

- End to end flows may be declared in a separate classifier extension
- No conflicting end to end flow declarations

```
System Top.flows extends top.basic
is
  Sensor_to_Actuator: end to end flow sensor1.reading -> … -> actuator1.cmd;
End;

Configuration Top.config_full extends Top.config_L2, Top.flows end;
```

- Flow specs for end-to-end flow targets may be declared in separate configurations
- Flow implementations for intermediate flow targets may be declared in a separate configurations

```
configuration X.flowspec extends X
is
  outsource: flow source outp;
End ;
configuration X.flowsequence extends x.i
is
  outsource => flow subsub1.flowsrc -> … -> outp;
End;
```

Configuration/composition of Annex Subclauses

Adding in annex specifications

- Annex subclauses may be declared in a separate classifier extensions
- Different annex specifications may be added

```
System Top_emv2 extends top is
Annex EMV2 {**
  use types ErrorLibrary;
  …
**};
End Top_emv2;
```

```
subclause Top_emv2 for top
use types ErrorLibrary;
  …
End Top_emv2;
```

Example of separately stored annex subclause

```
Configuration Top.config_full extends Top.config_L2, Top.flows, Top_emv2 end;
```

Inherited annex subclauses based on **extends**

- Automatically included
- Extends override rules of annex apply

Separate extensions

- No conflicting declarations

New idea: mode specific configuration specification: for property assignment.

Parameterized Configuration

Explicit specification of all choice points

- Configuration of subcomponents via configuration parameters only
  - Assignment of formal parameter to one or more subcomponents
- No direct configuration assignment to subcomponents by user
- Substitute the type of the parameter specification

```
Configuration x.configurable_dual(replicate: custom subsubsys) extends x.i is
  xsub1 => replicate;
  xsub2 => replicate;
end;
```

Configuration parameter classifier must the same or an ancestor of the assignment target

Similar to V2 prototype but we map parameter to targets instead of requiring all targets to reference prototype

Usage

- Supply parameter values

```
Configuration Top.config_sub1_sub2 extends top.i
is
  Sub1 => x.configurable_dual( replicate => subsubsys.i );
end;
Configuration x.configured extends x.configurable_dual( replicate => subsubsys.i )
end;
```

Configuration parameter actual must match
- an implementation/configuration of the specified interface
- a configuration of the specified implementation or its ancestor or interface

Explicit Specification of Candidates

- Explicit list of candidates

```
Configuration x.configurable_dual(securityProperties: system {
subsubsys.sec1, subsubsys.sec2 } ) extends x.i is
  xsub1 => securityProperties;
  xsub2 => securityProperties;
end;
```

Property Values as Parameters

Explicit specification of all values that can be supplied to properties

- Values that can be used for different properties of the same type
- Values for specific properties

```
Configuration x.configurable_dual(TaskPeriod : time ,
    TaskDeadline : #Deadline) extends x.i is  (need #Deadline? Limit value to assignment to deadline)
  xsub3.T1#Period => TaskPeriod;
  xsub3.T1#Deadline => TaskDeadline;
end;
```

Xsub2.T1 must exist in x.i

Usage: Supply parameter values

```
Configuration Top.config_sub1_sub2 extends top.i is
  Sub1 => x.configurable_dual(
    TaskPeriod => 20ms, TaskDeadline => 30 ms );
end;
```

Via configuration specification as parameter

- Collections of property value assignments
  - Consistent set of property values
- Explicitly specified collections to choose from

```
Configuration x.configurable_dual1(securityProperties: system subsubsys.i ) extends x.i is
  xsub1 => securityProperties;
  xsub2 => securityProperties;
end;
Configuration x.configurable_dual2(securityProperties: system { subsubsys.sec1, subsubsys.sec2 } )
extends x.i is
  xsub1 => securityProperties;
  xsub2 => securityProperties;
end;
```

Complete Configuration

- Finalizing choice points of an existing implementation or configuration

```
Configuration Top.config_L0() extends top.basic end;
```

- Users are able to add “missing annotations”
  - Additional flows, error model specification, property values
  - User can declare extensions of parameterized configuration that contain the annotations
  - User can compose multiple such annotations into the configuration
    - As new configuration or as part of each usage

```
Configuration Top.L0_Security extends Top.config_L0
is <security properties> end;
Configuration Top.L0_Safety extends Top.config_L0
is <EMV2 subclause for Top> end;
```

Configuration Assignment Patterns

Match&replace classifier/data type within a scope

- Match classifier in subcomponents and features, data types in features

```
Configuration FlightSystem.secure
    extends FlightSystem.TripleRedundant
is
  GPS *=> GPS.secure;
  Dlib::dt *=> Secure.securesample;

  #Period *=> 20 ms;
end;
```

Assign GPS.secure for all subcomponents with interface GPS within scope of FlightSystem.TripleRedundant

Assign type Secure.securesample for all features with type dt within scope of FlightSystem.TripleRedundant

Period for all elements within scope of associated implementation that require a Period

```
Package FS
Import mine::*;
System FlightSystem.TripleRedundant
is
 gps1: device GPS;
 gps2: device GPS;
 gps3: device GPS;
End;
End;
```

```
Package mine
Device interface GPS
is
  inp1: in data port Dlib::dt;
  outp1: out data port Dlib::dt;
End;
Device GPS.secure is
```

Generic Configuration Patterns

Match&replace within the scope the configuration pattern is assigned to

- Match classifier or primitive type in subcomponents and features
- Configuration without extends can be (Do I still need the implementaiton specific configuration pattern specification?)

```
Configuration GPSsecure.config is
  Mine::Sensor *=> Sensor.Settings;
  Dlib::dt *=> Secure.securesa
  #Period *=> 50 ms;

  Mine::GPS *=> GPS.secure { #Period => 50 ms};
end;
Configuration Sensor.Settings extends Sensor.impl is
  #Period => 50 ms;
  reading#Data_Size => 20 Bytes;
end;
```

Set Period default value within scope for any component requiring period and does not have an explicit assigned value

Assign period as part of pattern. Why not define classifier that includes the property

- Assign configuration pattern to subsystems

```
Configuration AvionicsSystem.Dual is
  FlightSystem1 => FlightSystem.primary, GPSsecure.config;
  FlightSystem2 => FlightSystem.primary, GPSsecure.config;
  BackupFlightSystem => FlightSystem.backup, SimpleGPS.config;
```