AADL Configuration Specification

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Architecture Design & Configuration

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

- Expand and restrict design choices in terms of architectural structure and other characteristics

System configuration

- Finalized choices of a given architecture design
- Composition of configuration specifications
- Parameterized configurations

Configuration of a System Design

Configuring subcomponents

- Any subcomponent is an implicit choice point
- Finalize subcomponent classifier
- Classifier as root of name paths

- Configuration of one level

```
configuration Top.config_L1 extends top.basic
(
Sub1 => x.i,
Sub2 => y.i
);
```

- Configuration of multiple levels

```
configuration Top.config_Sub1 extends top.basic
(
Sub1 => x.i,
Sub1.xsub1 => subsubsys.i,
Sub1.xsub2 => subsubsys.i
);
```

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

```
System implementation x.i
 Subcomponents
 xsub1: process subsubsys;
 xsub2: process subsubsys;
```

Syntax is similar to prototype actuals

Refinement rules apply
Classifier_Match, Type_Extension, Signature_Match

Nested Configuration Syntax

Configuring subcomponents several level down

- Configuration of multiple levels

```
configuration Top.config_Sub1 extends top.basic
(
  Sub1 => x.i(
    xsub1 => subsubsys.i,
    xsub2 => subsubsys.i
  )
);
```

Alternative to

```
Top.config_Sub1 configures top.basic
(
  Sub1 => x.i,
  Sub1.xsub1 => subsubsys.i,
  Sub1.xsub2 => subsubsys.i,
);
```

Use of Configurations in Configurations

Specification and use of separate subsystem configurations

- Configuration of subsystems

```
Configuration x.config_L1 extends x.i (
  xsub1 => subsubsys.i,
  xsub2 => subsubsys.i
);
Configuration y.config_L1 extends y.i (
  ysub1 => subsubsys.i,
  ysub2 => subsubsys2.i
);
```

- Use of subconfigurations

```
Configuration Top.config_L2 extends top.i (
  Sub1 => x.config_L1,
  Sub2 => y.config_L1
);
Configuration Top.config_L1Sub1 extends top.L1 (
  Sub1 => x.config_L1
);
```

Independent of top.conf_L1

Override of a classifier by a configuration of the classifier

```
configuration Top.config_L1
   extends top.basic
(
Sub1 => x.i,
Sub2 => y.i
);
```

Configuration of Property Values

Finalizing a set of property values

- These are the values that apply to the instance model
  - Overrides previously assigned values and cannot be overriden
- Only for model elements whose presence cannot be changed
  - Legality of subcomponent path determined by referenced classifier

```
Configuration Top.config_Security extends Top.config_L1Sub1
(
  #Security_Level => L1,
  Sub1#Security_Level => L2,
  Sub1.xsub1#Security_Level => L0,
  Sub2#Security_Level => L1
);

Configuration Top.config_Safety extends Top.config_L1
(
  #myps::Safety_Level => Critical,
  Sub1#myps::Safety_Level => NonCritical,
  Sub2#myps::Safety_Level => Critical
);
Configuration x.config_Performance extends x.i
(
  xsub1 =>(
   #Period => 10ms,
   #Deadline => 10ms )
);
```

A configuration specification with only property associations acts like a data set that applies to a design.
It can be combined with others through configuration composition.

Reference to elements in implementation.
Name paths continue to be valid as long as we prohibit refined to of subcomponents where an implementation is replaced by another implementation that is not an extension of the one being replaced.

Previously Configured Subcomponents

Configuration of previously configured subcomponent

- We configure parts of a configured subcomponent that have not been previously configured

```
Configuration Top.config_Sub1 extends top.config_L1
(
  Sub1 => (
    xsub1 => subsubsys.i,
    xsub2 => subsubsys.i
  )
);
```

Expand Sub1 one level

```
Configuration Top.config_Sub2 extends top.config_L1
(
  Sub2 => y.config_L1
);
```

Expand Sub2 one level

Combine the two expansions

```
Configuration Top.config_full extends top.config_L1, Top.config_Sub1,
Top.config_Sub2;
```

Composition of Configurations

Combine structural configuration with different “data sets”

- Extends references are processed in order
- Configurations must reference configurations items in the extends hierarchy of a predecessor element

```
Configuration Top.config_full extends Top.config_L1Sub1 with
  Top.config_Safety,
  Top.config_Security
;
```

```
Configuration Top.config_SafetySecurity extends Top.config_Security with
Top.config_Safety;
```

Ok as safety references `Top.config_L1`

```
Configuration Top.config_SafetySecurity extends Top.config_Safety with
Top.config_Security;
```

Not ok, as security references `Top.config_L1Sub1`

Name Path Based Composition

Allow application of configurations as long as name paths match

- Configurations do not reference configurations items in the extends hierarchy of a predecessor element

```
Configuration Top.config_full extends Top.config_Sub1,
  unsafe Top.config_Safety,
  unsafe Top.config_Security
;
```

`Top.config_L1` of `Top.config_Safety` is not in the extends hierarchy of `Top.config_Sub1.`
However, the subcomponent name paths are in `Top.config_Sub1.`

Parameterized Configuration

Explicit specification of all choice points

- Only the choice points can be configured by users
- No direct external configuration of elements inside

Explicit specification of where choice points are used

- Choice point can be used in multiple places

```
Configuration x.configurable_dual(replicate: system subsubsys) extends x.i
(
  xsub1 => replicate,
  xsub2 => replicate
);
```

Refinement substitution rules apply to application of choice point.

Usage

- Supply parameter values

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

Refinement substitution rules apply to supplied choice point actual.

Property Values as Parameters

Explicit specification of all values that can be supplied to properties

- **Only choice point property values can be configured**
- **Choice point can be used in multiple places**

```
Configuration x.configurable_dual(replicate: system subsubsys,
    TaskPeriod : time) extends x.i (
  xsub1 => replicate,
  xsub2 => replicate,
  xsub1#Period => TaskPeriod,
  xsub2#Period => TaskPeriod
);
```

Usage: Supply parameter values

```
Configuration Top.config_sub1_sub2 extends top.i (
  Sub1 => x.configurable_dual(
    replicate => subsubsys.i,
    TaskPeriod => 20ms
  )
);
```

Parameterized Configuration

Match&replace within a scope

- Match classifier in subcomponents and features
- Match property name
- Recursive
- Scoped

```
System x
 Features
  inp1: in data port Dlib::dt;
 outp1: out data port Dlib::dt;
```

```
Configuration x.configurable_dual(replicate: system subsubsys,
    streamtype: data Dlib::dt,
    TaskPeriod : time) extends x.i
(
  * => replicate,
  *#Period => TaskPeriod,
  xsub1.* => streamtype,
  xsub1.*#Deadline => TaskPeriod
);
```

Replace matching subsubsys classifier

Set period where Period is accepted

Match data classifier within xsub1 subtree

Explicit Specification of Candidates

Default: all classifiers according to matching rules

Explicit: Candidate list

```
Configuration x.configurable_dual(
replicate: system subsubsys{subsubsys.i, subsubsys.i2}
   ) extends x.i
(
  xsub1 => replicate,
  xsub2 => replicate
);
```

Complete Configuration

- Finalizing an existing implementation or configuration without change

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

Nested Configurable Systems: An Example

Sound system inside the entertainment system is closed

- Speaker selection as choice point

```
System implementation MySoundSystem.design
Subcomponents
  amplifier:  system Amplifier.Kenwood;
  speakers: system Sound::Speakers;
End MySoundSystem.design;

Configuration MySoundSystem.Selectablespeakers (speakers: system
Sound::Speakers) extends MySoundSystem.design
(  speakers => speakers );
```

Entertainment system is open design

```
System implementation EntertainmentSystem.basic
Subcomponents
  tuner:  system Tuner.Alpine;
  soundsystem: system MySoundSystem.Selectablespeakers;
End EntertainmentSystem.basic;
```

Nested Configurable Systems - 2

PowerTrain with choice of engine

- Gas engine choice as only choice point

```
System implementation Powertrain.design
Subcomponents
  myengine:  system EnginePkg::gasengine;
End Powertrain.design;

Configuration PowerTrain_gas (gasengine : system EnginePkg::gasengine)
extends Powertrain.design
( myengine => gasengine;
);
```

Nested Configurable Systems - 3

All choice points as top level parameters

- Parameters are mapped across multiple levels for speaker selection

```
System implementation car.design
Subcomponents
  PowerTrain:  system PowerTrain.gas ;
  EntertainmentSystem:  system EntertainmentSystem.basic;
End car.configurable;

Configuration car.configurable (g_engine: system Pckg::gasengine ,
speakers: system Sound::Speakers ) extends car.design
PowerTrain.g_engine => g_engine ,
EntertainmentSystem.Soundsystem.speakers => speakers
);

Configuration car.config extends car.configurable
( gasengine => Pckg::engine.V4 , speakers => Custom::Speakers.Bose)
End car.config;
```

Refinement Rules

For prototypes – same as for classifier refinement (V2)

- Always: no classifier -> classifier of specified category.
- Classifier_Match: The component type of the refinement must be identical to the component type of the classifier being refined. Allows for replacement of a “default” implementation by another of the same type. [Nothing changes in the interfaces]
- Type_Extension: Any component classifier whose component type is an extension of the component type of the classifier in the subcomponent being refined is an acceptable substitute. [Potential expansion of features within extends hierarchy]
- Signature_Match: The actual must match the signature of the prototype. Signature match is name match of features with identical category and direction
  - Actual with superset of features in type extension or signature: results in unconnected features that must be connected in design extensions
  - Not allowed for configurations
  - Need for order matching (allows for different feature names)
  - Need for name mapping of features when actual is provided? (VHDL supports that)
  - We provide name mapping for modes to requires modes