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

- Selections for choicepoints of a given architecture design
- Composition of configuration specifications
- Parameterized configurations

Configuration of a System Design

Configuring subcomponents

- Any subcomponent in the hierarchy is a choice point
- Select component implementation for subcomponents
  - Their elements may still be choice points with just a type
- Associate “annotations” to an architecture design such as property values, bindings, annexes
  - Model elements being annotated do not change

Configuration of one level

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

Replacement of type by implementation

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

Configuration Across Multiple Levels

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

```
configuration Top.config_Sub1 extends top.sub1impl
{
  Sub1.xsub1 => subsubsys.i,
  Sub1.xsub2 => subsubsys.i
};
```

```
System implementation top.sub1impl
 Subcomponents
 Sub1: system x.i;
 Sub2: system y;
```

- Nested configuration assignments
  - Used when configuring an assigned classifier
  - Left hand side resolved relative to enclosing assigned classifier

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

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 configuration as assignment value

```
Configuration Top.config_L2 extends top.basic {
  Sub1 => x.config_L1,
  Sub2 => y.config_L1
};
```

Previously Configured Subcomponents

Configuration of previously configured subcomponent

- Configuring subcomponents in configurations by reach down

```
Configuration Top.config_L2 extends top.config_L1 {
  Sub1.xsub1 => subsubsys.i,
  Sub1.xsub2 => subsubsys.i,
  Sub2 => { ysub1 => subsubsys.i ,
            ysub2 => subsubsys.i
          }
};
```

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

- Configuration by replacing a previously assigned implementation by an extension of the implementation

```
Configuration Top.config_Sub2 extends top.config_L1
{
  Sub1 => x.config_L1
  Sub2 => y.security
};
```

```
System implementation y.security
extends y.i
 properties
 <security properties>
```

Replacement of an implementation by a configuration of the implementation

Replacement of an implementation by a extension of the implementation that contains properties or refinements

Extensions that contain flows, annex subclauses, additional subcomponents, connections

Configuration of Property Values

Finalizing a set of property values

- Property value assignment to any component in the
  - subcomponent path resolvable via the classifier referenced by **extends**
  - May override previously assigned values and cannot be overwritten

```
Configuration Top.config_Security extends Top.config_L2
{
  #myps::Security_Level => L1,
  Sub1#myps::Security_Level => L2,
  Sub1.xsub1#myps::Security_Level => L0,
  Sub2#myps::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 => x.i {
   #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.

Equivalent to myps::security_level => L2 applies to Sub1
We will use the same property association syntax consistently.
Consistent with reference syntax used in BA

Composition of Configurations

Combine structural configuration with different extensions that represent different annotations/”data sets”

- Extends reference identifies configuration or component classifier to be augmented
- Ensure that all model element references being composed remain valid

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

Configuration Top.config_full extends Top.config_L2, Top.config_Safety,
  Top.config_Security
;

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

Order in extends list is not relevant.
All elements must be in the same extends hierarchy.

Multiple configurations assign property value:
- Extension property value is assigned
- Parallel extensions with different values is conflict

Unnamed composition as part of a subcomponent configuration.

```
Configuration Top.config_L2 extends
top.basic {
  Sub1 => x.config_L1, x.security; -- comma
the right symbol?
  Sub2 => y.config_L1
};
```

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
};
```

Configuration assignment 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 )
};
```

Configuration assignment substitution rules apply to the 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
};
```

No “section” markers to separate classifier assignment and property associations.

Usage: Supply parameter values

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

Parameter Match and Replace

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, tasktype: thread Tlib::task,
    TaskPeriod : time) extends x.i
{
  * => replicate,
  *#Period => TaskPeriod,
  xsub1.*: => tasktype,
  *.outp => streamtype,
  xsub1.*#Deadline => TaskPeriod
};
```

Replace matching subsubsys classifier

Set period where Period is accepted

Match data classifier within xsub1 subtree

Match data classifier for all matching port names

Set all subcomponent deadlines within xsub1 to the task period parameter value

Explicitly assigned property value takes precedence over match&replace

Multiple patterns for same replacement: more specific pattern applies
Same match with different replacements: error

Support match&replace in implementation **refined to** and property assignment?

Explicit Specification of Candidates

Default: all classifiers according to matching rules

Explicit: Candidate list

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

Complete Configuration

- Finalizing an existing implementation or configuration without any choice points

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

Parameterized Configurations and Annotations

- Model may be missing EMV2 specification, security properties, or project specific properties
  - 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.config_L0() extends top.basic;

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

Configuration of Annex Subclauses

Adding in annex specifications

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

```
System Top_emv2 extends top
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 with Top.flows, Top_emv2 ;
```

Inherited annex subclauses based on **extends**

- Automatically included

Composition of Flow Configurations

Adding in end to end flows

- End to end flows may be declared in a separate classifier extension

Someone else's model does not include flows of interest. Can we add them in without modifying the original model?

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

Configuration Top.config_full extends Top.config_L2 with Top.flows ;
```

- Flow specs may be declared in a separate type extension
- Flow implementations may be declared in a separate implementation extension

```
System X_flows extends X
Flows
  outsource: flow source outp;
End X_flows;
System implementation X_Flows.flows extends x.i
Flows
  outsource: flow source subsub1.flowsrc -> … -> outp;
End X_Flows.flows;
```

Do we need to specify both the flow spec and flow implementation ?

Alignment with Core V2 Syntax

Configuration vs. implementation

- **Refined to** limited to *type_extension*
- Bracketing {} vs. “end”, “section” keywords

=> vs. refined to

- Syntax alternatives

Extends multiple

- Allow for implementations too
  - Addition of subcomponents, flows, connections
- Composition rules for configurations apply to implementations

Do we need to distinguish between type match, *type_extension*, and subset? Seems like we always want to allow type extension and handle subset via interfaces

Configuration parameter vs. prototype

- Prototype requires reference of prototype throughout model, but only directly in implementation
  - Requires repetition of prototype down the hierarchy
- Configuration parameter specifies mapping of parameter into the model across multiple levels
  - Allows for parameterization of existing model without modification

Property association with and without **applies to**

- **Sub1#period => 20 ms;**

Alignment with Core V2 Syntax Example

```
system Top.config_Safety extends Top.config_L1
properties
  #myps::Safety_Level => Critical;
  Sub1#myps::Safety_Level => NonCritical;
  Sub2#myps::Safety_Level => Critical;
End Top.config_Safety;

System Top.config_L2 extends top.config_L1
subcomponents
  Sub1.xsub1 : refined to subsubsys.i;
  Sub1.xsub2 : refined to subsubsys.i;
 Sub2.ysub1 : refined to subsubsys.i;
 Sub2.ysub2 : refined to subsubsys.i;
end;
```

```
System Top.config_L1 extends top.basic
subcomponents
Sub1 : refined to system x.i;
Sub2 : refined to y.i;
end;
```

Semantic differences:
Property values cannot be overwritten once configured?
Refined to with signature match?

Multiplicities (Arrays)

V3 support

- Configuration of dimensions

```
System implementation top.design
subcomponents
Sub1 : system S[];
Sub2 : system S[];

top.config configures top.design
( Sub1 => [10] , Sub2 => S.impl[15]);
```

Multiplicities Reflected in Features

V3 support

- Configuration of dimensions

```
System top
Features outp: out data port[2][];

System implementation top.design
subcomponents
Sub1 : system S[];
Sub2 : system S[];
connections
C1: port Sub2.outport -> outp[1][];
C2: port Sub2.outport -> outp[2][];

top.config(copies: integer 2..10) configures top.design
( outp => [][copies],Sub1 => [copies] , Sub2 => S.impl[copies]);
```

Indication that the port will carry an array and not force a fan-in

Acceptable values within range
Request for power of 2:
2^(2..10)

Internal subcomponent arrays mapped into feature array

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);
```

V2.2 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