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
};
```

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.

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

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
  )
};
```

Complete Configuration

- Finalizing choice points of an existing implementation or configuration

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

- 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.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 Previously Configured Subcomponents

Existing topology does not change but may be expanded

- 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
          }
};
```

Add configuration rather than replace configuration

```
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_sec extends top.config_L1
{
  Sub1 => x.config_L1
  Sub2 => y.security
};
```

Replacement of an implementation by a configuration of the implementation

```
Configuration y.security extends y.i
 properties
 <security properties>
```

Replacement of an implementation by a extension of the implementation that contains properties or refinements BUT not addition (equivalent to configuration)

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

Extensions that contain flows, annex subclauses

Extensions that add subcomponents, connections
- Ok in implementations
- it changes topology for configurations

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**
  - May override previously assigned values

```
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
};
```

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.

```
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 => subsubsys.i {
   #Period => 10ms,
   #Deadline => 10ms }
};
```

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 multiple configurations into new configuration

- Ensure that topology does not change but may be expanded
  - Ensures that existing model element references remain valid
- Additional configurations are extensions of first

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

Other elements must be extensions of first

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

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

```
Configuration Top.config_SafeSecure extends Top.config_L2 with Top.config_Safety,
  Top.config_Security ;
```

We just add property values to Top.config_L2
Other elements must be extensions of first or one of its super types

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

Infer highest level structural configuration to be Top.config_L2

Composition Rules

Configuration assignments

- Additional extensions make non-overlapping assignments
  - Configuration expansion for different subsystems
  - Different sets of property values (safety, security)
  - Flows, annex declarations
- Overlapping configuration assignments
  - One is extension of other configuration => use extension
  - Extensions from same root: same structural configuration or subset => use superset
- Overlapping property assignments
  - One is extension of the other => extension value takes precedence
    - same as local assignment in an extension
  - Two separate extensions from same root: no conflicting values

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
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
- Extends override rules of annex apply

Separate extensions

- No conflicting declarations

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 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 ?

Unnamed Compositions

Unnamed composition as part of a subcomponent configuration

- Do we need to support this or require composite configurations to be defined before use

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

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

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?

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