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)

- 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
- As extension of component implementation, configuration, interface

- Configuration assignments
  - implementation to subcomponents
  - Types or classifier to features
  - Association of collections of final property values, bindings, annexes to given architecture substructure via configuration
- *Final* property values, binding, annexes declarations as classifier extensions

Composition of configuration specifications

Parameterized configuration specification

- Subcomponent configuration assignment via parameter only

Evolution of System Design

Component Interface Extension

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

Component Implementation Extension

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

V2 type match and type extension

Eliminated signature match and need for substitution rule specification

Configuration of a System Design

Configuration Specification

- Associated with an implementation (identified by extends)

Configuration assignment

- Elaborate and annotate subcomponent substructure
  - Select component implementation for subcomponent with interface
    - Must be same as existing implementation if present
  - Annotate substructure with “final” property values, bindings, annex subclauses
  - Assign configuration for existing subcomponent implementation

Configuration of one level

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

Replacement of interface by implementation

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

Configuration Across Multiple Levels

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

```
configuration Top.config_Sub11 extends top.sub1impl
is
  Sub1.xsub1 => subsubsys.i;
  Sub1.xsub2 => subsubsys.i;
end;
```

```
System top.sub1impl is
 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
is
  Sub1 => x.i {
    xsub1 => subsubsys.i;
    xsub2 => subsubsys.i;
  }
end;
```

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

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

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.Config_L1 is
  Sub1 => x.config_L1;
  Sub2 => y.config_L1;
end;
```

Implementation associated with configuration must be the same or an ancestor of previously assigned 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**
  - May override previously assigned 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 with only property associations is an annotation of a fixed architecture structure.

Composition of Configurations

Combine multiple configurations into new configuration

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

Rules

- Associated implementations must be the same
  - For configuration assignment also the same as the existing implementation of the subcomponent
- 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 configuration composition

```
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;
```

Configuration Assignment Patterns

Match&replace classifier within a scope

- Match classifier in subcomponents and features

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

```
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;
End;
```

Generic Configuration Patterns

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

- Match classifier or primitive type in subcomponents and features
  - * is confusing. Universal configuration

```
Configuration *.GPSsecure is
 Mine.GPS => GPS.secure { #Period => 50 ms;};
 Mine.Actuator => {#Period => 50 ms;};
 Mine.Sensor => Sensor.Settings;
 Dlib.dt => Secure.securesample;
end;
Configuration Sensor.Settings extends Sensor.impl is
  #Period => 50 ms;
  reading#Data_Size => 20 Bytes;
end;
```

- Assign configuration pattern to subsystems

```
Configuration AvionicsSystem.Dual is
  FlightSystem1 => *.GPSsecure;
  FlightSystem2 => *.GPSsecure;
  BackupFlightSystem => *.SimpleGPS;
```

Wildcard Configuration Patterns

```
Configuration *.unconfigured is
  *.inport => datatype ;
  *#Period => 30 ms;
end;
```

Set type for all matching port names and no type

Set Period where Period is accepted and not final

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

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

Configuration parameter classifier must the same or an ancestor of the assignment target
Arbitrary substitution for components without features?
Subcomponent without classifier

Usage

- Supply parameter values

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

```
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 actual must be an implementation of the interface of the parameter targets, or a configuration of an implementation whose interface is that of the target interface and is the same or an ancestor of the target implementation (if preent)

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(replicate: system subsubsys,
    TaskPeriod : time , TaskDeadline : #Deadline) extends x.i is
  xsub1 => replicate;
  xsub2 => replicate;
  xsub3.T1#Period => TaskPeriod;
  xsub3.T1#Deadline => TaskDeadline;
end;
```

‘replicate’ may be a configuration that includes property associations

Xsub2.T1 must exist in x.i

Usage: Supply parameter values

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

Parameters in Configuration Patterns

```
Configuration x.configurable_dual(replicate: system subsubsys,
    streamtype: type, tasktype: thread Tlib.task,
    TaskPeriod : time) extends x.i is
  subsubsys => replicate;
  * => replicate;
  *#Period => TaskPeriod;
  xsub1 => {Tlib.task => tasktype;};
  * => streamtype;
End;
```

Replace matching subsubsys classifier

Configure subcomponents without classifier

Set Period where Period is accepted and not set

Configure threads with ‘Tlib.task’ within xsub1

Configure streamtype for any port/data access or data component without type

```
Configuration x.ConfigurableThreads(processConfig: process proc.impl
from { proc.storageconfig, proc.timingconfig, proc.securityconfig} )
extends x.i is
  proc.impl => processConfig;
end;
```

Supply configuration for process implementation from provided candidates

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.config_L0() extends top.basic;

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/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

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

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

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

“Graphical” representation for specifying configurations. Do we have a graphical symbol or recommend it be done by dialog.