Configuration of System Designs

Peter Feiler
Jan 2017

Things to Configure

Configuration of architecture structure

- Subcomponent type -> implementation

Feature classifiers

- Port data types
- Access types

Property values

Multiplicities/Arrays

Resource bindings

- Processor, memory, network, function

In modes configurations

Architecture Design & Configuration

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

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

System configuration to finalize choices of a given architecture design

- They can be used where classifiers are allowed
- Composition of configuration specifications
- Parameterizable configurations

Configuration of a System Design

Configuring subcomponents

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

```
Top.config_L1 configures top.basic
(
Sub1 => x.i, -- unchangeable assignment
Sub2 => y.i
);
```

- Configuration of multiple levels

```
Top.config_Sub1 configures 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;
```

Refinement rules apply
Classifier_Match, Type_Extension, Signature_Match

Compositional Configuration

Specification and use of separate subsystem configurations

- Configuration of subsystems

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

- Composition of level configurations

```
Top.config_L2 configures top.i (
  Sub1 => x.config_L1,
  Sub2 => y.config_L1
);
Top.config_L2 configures Top.config_L1 (
  Sub1 => x.config_L1,
  Sub2 => y.config_L1
);
```

Previously Configured Subcomponents

Configuration of previously configured subcomponent

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

```
Top.config_Sub1 configures top.config_L1
(
  Sub1 => (
    xsub1 => subsubsys.i,
    xsub2 => subsubsys.i
  )
);
Top.config_Sub1C configures top.config_L1
(
  Sub1 => x.config_L1
);
```

Nested Configuration Syntax

Configuring subcomponents several level down

- Configuration of multiple levels

```
Top.config_Sub1 configures 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,
);
```

Configuration of Feature Type

Configuring feature type

- For subcomponent

```
Top.config_Sub1 configures top.basic
(
  Sub1 => x.i,
  Sub1.inp1 => Dlib::dt1,
  Sub1.outp1 => Dlib::dt2
);
```

- In component implementation

```
x.config_L1 configures x.i
(
  inp1 => Dlib::dt1,
  outp1 => Dlib::dt2,
  xsub1 => subsubsys.i,
  xsub2 => subsubsys2.i
);
```

- In component type

```
X_pconfig configures x
(
  inp1 => Dlib::dt1,
  outp1 => Dlib::dt2
);
```

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

Configuration of Property Values

Finalizing a set of property values

- The value cannot be changed
- Only for model elements whose presence cannot be changed

```
Top.config_Security configures Top.config_Sub1
(
  #Security_Level => L1,
  Sub1#Security_Level => L2,
  Sub1.xsub1#Security_Level => L0,
  Sub2#Security_Level => L1
);

Top.config_Safety configures Top.config_Sub1
(
  #Safety_Level => Critical,
  Sub1#Safety_Level => NonCritical,
  Sub2#Safety_Level => Critical
);

x.config_Performance configures x.i
(
  #Period => 10ms,
  #Deadline => 10ms
);
```

A configuration specification with only property associations acts like a data set that applies to a design.

An interface with only property associations acts as a data sets that applies to a component (type and its features).

Composition of Configurations

Combine structural configuration with different “data sets”

- The additional configurations must be for the same “final” architecture design
- Only for model elements whose presence cannot be changed

```
Top.config_full configures Top.config_Sub1 with
  Top.config_Safety,
  Top.config_Security
(
  Sub1 => x.config_Performance
);

Top.config_SafetySecurity configures Top.config_Sub1, Top.config_Security
(
  #Safety_Level => Critical,
  Sub1#Safety_Level => NonCritical,
  Sub2#Safety_Level => Critical
);
```

Add in Interface with property associations only.

Parameterized Configuration

Explicit specification of all choice points

- Only choice points can be configured
- No direct external access to elements inside

Explicit specification of where choice points are used

- Choice point can be used in multiple places

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

Substitution rules apply
Classifier_Match, Type_Extension, Signature_Match

Usage

- Supply parameter values

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

Property Values as Parameters

Explicit specification of all values that can be supplied to properties

- Only choice points can be configured
- Choice point can be used in multiple places

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

Substitution rules apply
Classifier_Match, Type_Extension, Signature_Match

Usage: Supply parameter values

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

Explicit Specification of Candidates

Default: all classifiers according to matching rules

Explicit: Candidate list

```
x.configurable_dual(
replicate: system subsubsys{subsubsys.i, subsubsys2.i}
   ) configures x.i
(
  xsub1 => replicate,
  xsub2 => replicate
);
```

Complete Configuration

- Finalizing an existing design without change

```
Top.config_L0() configures top.basic;
```

- Finalizing an existing design without change

```
Top.config_L0() configures top.basic;
```

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;
```

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

Replace matching subsubsys **classifier**

Set period where Period is accepted

Match data classifier within xsub1

Nested Configurable Systems

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;

MySoundSystem.Selectablespeakers (speakers: system Sound::Speakers)
configures 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;

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

Nested Configurable Systems - 2

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;

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

car.config configures 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

Feature name mapping in Signature match

```
Abstract controller
Features
Input: in data port;
Output: out data port;
End controller;

System brakecontroller
Features
speedreading: in data port;
brakeactuationsignal: out data port;
End controller;

brakesystem.config (controller: system pck::Controller, sensor: system pck::sensor)
configures breakesystem.design
End brakesystem.config;

System implementation aircraft.system
Subcomponents
  abs:  system brakesystem.config( sensor => pck::speedsensor, controller =>
brakecontroller(input = speedreading, output = brakeactuationsignal));
End aircraft.system;
```

= for feature name mapping

Array Sizes

V2 support

- Refined to of subcomponent/feature
- Use of property constants
  - Property constants are global within workspace
- Scoped “constants” aka. Prototypes for array size

- Acceptable range of values.

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

Variability Points

Configuration of architecture structure

Feature classifiers

Array sizes

Refined to revisited

Final property values

Resource bindings: bindings proposal

Extends and Refined to

Architecture design via extends and refines (V2)

One layer at a time

- Addition of new and refinement of existing model elements
- In component types
  - Add and refine interface features
  - Override property values
- Component implementations
  - Add and refine subcomponents
  - Override property values including binding specifications

Prototype & prototype actual (V2)

One layer at a time

- Classifiers for features of component types
- Classifiers for subcomponents of implementations

V3 Proposal

Specify selection multiple levels down in architecture design

- As part of refined to

```
Sub1.sub11.sub112 : refined to system system.Implementationx ;
```

- As part of classifier refinement with choice actual

```
Sub1.sub4: refined to system gps.impl(sensorproto => sensor.i);
```

Refinement of Architecture Design

Ability to refine across multiple architecture levels

- Any subcomponent can be refined according to refinement rules
- Reachdown reduces need for classifier extensions of intermediate levels

```
System implementation top.basic
Subcomponents
  Sub1:  system subsys;
  Sub2:  system othersys;
End top.basic;

System implementation subsys.basic
Subcomponents
  Subsub1:  system subsubsys;
End subsys.basic;

System implementation top.refined extends top.basic
subcomponents
  Sub1 : refined to system Subsys.i;
-- refine an element of the subsystem just refined
  Sub1.subsub1 : refined to system subsubsys.i;
```

Need for Prototype and Refined To

Prototype

- Within design space indicate that the same classifier is to be used in multiple places
  - Configuration parameter achieves the same thing

Refined to

- Further constrain subcomponent type by subtype
- Configure in implementation
- Override implementation as a way of configuring in a more detailed implementation

Extends

- Can be limited to extensions if refined to is a configuration without need for override

Default and final property values

"final" property associations

- Currently
  - default in definition
  - Changeable unless declared "constant"
  - Can be overwritten locally and by "applies to"
- Proposal
  - No more default in definition
    - Context specific default:
      - component type,
      - enclosing component and **inherit**

Inheritance & Overriding of Property Values

final as default for =>

- Assign once only
- Changeable in extends and implementation
  - Special syntax for value assignment

Default value for properties at definition time

- Examine each for actual need
- Alternatives
  - Scoped default with component classifier or enclosing component

Inherit from enclosing component

- Still needed if we have prototype parameterization of property values