Configuration of Variability Points

Peter Feiler
July 2016

Variability Points

- Configuration of architecture structure
  - Subcomponent type -> implementation
- Feature classifiers
  - Port data types
  - Access types
- Multiplicities/Arrays
- Property values
- Resource bindings
  - Processor, memory, network, function
- In modes configurations

Architecture Design & Configuration

Architecture design via extends, refines, prototype to evolve design space

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

System configuration to finalize selections for variability points of a chosen architecture design

Architecture Design

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 **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 prototype actual

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

- As part of prototype usage

```
Sub1.sub3.sensorproto => sensor.i;
```

Architecture Design and Parameterized Configuration

- Architecture design allows reach down into subcomponents
- Parameterized configuration has all variability points as prototypes
  - No reach down from outside

Refinement of Architecture Design

Ability to refine across multiple architecture levels

- 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;
```

Refinement of Architecture Design

Dealing with prototypes

- No need to reach beyond prototype

```
System implementation subsys.proto
Prototypes
  subsub: system subsub;
Subcomponents
  Subsub1:  system subsub;
End subsys.basic;

System implementation top.refined extends top.basic
subcomponents
  Sub1 : refined to system Subsys.i (subsub => subsubsys.i);

System implementation top.refined extends top.basic
subcomponents
  Sub1 : refined to system Subsys.i ;
  Sub1.subsub => subsubsys.i;
```

Without prototype actual in brackets

Configuration of a System Design

We are responsible for several levels of the component hierarchy

- We configure implementations
  - Once configured they cannot change
- Selections may be configurable themselves

```
System configuration top.config1 extends top.basic
(
Sub1 => Subsys.proto; -- unchangeable assignment
Sub1.subsub => subsubsys.i;
)
```

Prototype actual style syntax

- Configurations may be partial, i.e., require additional selections

```
System configuration top.partconfig extends top.basic
( Sub1 => Subsys.i )

System configuration top.fullconfig extends top.partconfig
( Sub1.subsub1 => subsubsys.i );
```

Configuration of a System Design

We are responsible for several levels of the component hierarchy

- We configure implementations
  - Once configured they cannot change
- Selections may be configurable themselves

```
System configuration top.config1 extends top.basic
subcomponents
Sub1 : refined to system Subsys.i;
Sub1.subsub1 : refined to system subsubsys.i;
```

Implementation refinement style

We will use it also for feature configuration

Features can only be refined in types

Parameterized System Configuration

```
System implementation car.basic
Subcomponents
  MyPowerTrain:  system PowerTrain;
  MyEntertainmentSystem:  system EntertainmentSystem.basic;
End car.basic;

System configuration PowerTrain.gas extends Powertrain.design
Prototypes
 -- gasengine : system gasengine;
 -- gasengine : system;
    system gasengine;
End PowerTrain.gas;
System implementation Powertrain.design
Subcomponents
  myengine:  system gasengine;

System configuration car.config extends car.basic
( MyPowerTrain => PowerTrain.gas (gasengine =>  gasengine.V4))
-- OR
( MyPowerTrain => PowerTrain.gas,
  MyPowerTrain.gasengine =>  gasengine.V4)
```

Classifier acts as prototype name

We must have a classifier in subcomponent because once configured we cannot add connections etc. for subcomponent.

Need for nested brackets?

Refined to are not nested

Parameterized System Configuration

```
System implementation car.basic
Subcomponents
  MyPowerTrain:  system PowerTrain.gas;
  MyEntertainmentSystem:  system EntertainmentSystem.basic;
End car.basic;

System configuration car.configurable extends car.basic
Prototype
  system gasengine ;
End car.configurable;

System configuration car.config extends car.basic
( MyPowerTrain.gasengine => gasengine.V4 )
```

Provide prototype actual

V2 Support for Feature Classifier

```
System s
Prototypes
   bustype: bus bustype;
features
  p1:  in data port;
  p2:  requires bus access bustype;
End s;

System SRefined extends S (bustype => bus Ethernet)
features
  p1: refined to in data port dt;
End SRefined;

System implementation top.basic
Subcomponents
 Sub1:  system S (bustype => bus Ethernet);
 Sub2:  system S.basic (bustype => bus Ethernet);
End subsys.basic;
```

V3 Support for Feature Classifiers

```
System S1
features
  p1:  in data port dt;
  p2:  requires bus access bustype;
End S1;
System S2
features
  p1:  out data port dt;
  p2:  requires bus access bustype;
End S2;

System configuration top.config of top.design
Prototypes
  data dt;
End top.config;
System implementation top.design
subcomponents
Sub1 : system S1;
Sub2 : system S2;
End top.design;
```

Match prototype classifier name with classifiers

Default: all matching

Nested Configurable Systems

Sound system inside the entertainment system is closed

- Speaker selection as variability point

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

System configuration MySoundSystem.Selectablespeakers extends
MySoundSystem.design
Prototypes
  system speakers;
End MySoundSystem.Selectablespeakers;

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

Classifier prototype will be matched by assignment in next slide

Nested Configurable Systems - 2

All variability points as top level prototypes

- Prototypes are mapped across multiple levels (speaker selection)

```
System configuration car.configurable extends car.design
Prototypes
  system gasengine;
  system speakers;

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

System configuration car.config extends car.configurable
( gasengine => engine.V4 , speakers => Speakers.Bose)
End car.config;
```

Scope of Application for Prototype

Default: all matching classifiers in subcomponents

- Recursive (reach down) for system implementations

Explicit target specification of configuration parameter (prototype)

```
System implementation Powertrain.design
Subcomponents
 mainengine:  system gasengine.i;
 backupengine:  system pkge::electricengine;
End design;
System configuration PowerTrain.gas extends Powertrain.design
Prototypes
 [<alias>:] system mainengine.pkg::gasengine;  -- qualified name for
classifier
 System pcka::gasengine applies to mainengine
 system backupengine.electricengine;
End PowerTrain.gas;
```

Identify root of scope and classifier

Scoped configuration of Feature Classifiers

```
System configuration top.config extends top.design
Prototypes
  data Sub1.dt;
  data Sub2.p1.*;
End top.config;
```

Scope root is subcomponent (or port)

Identify port if port does not have classifier and use * for any

Explicit Specification of Candidates

Default: all classifiers according to matching rules

Explicit:

Option 1: Candidate list in syntax

```
System configuration PowerTrain.gas extends Powertrain.design
Prototypes
  system gasengine choices gasengine.v4, gasengine.V6;
  system electricengine;
End PowerTrain.gas;
```

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 one 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;

System configuration brakesystem.config extends breakesystem.design
Prototypes
  system controller;
  system sensor;
End brakesystem.config;

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

= for feature name mapping

Without nesting

Variability Points

Configuration of architecture structure

Feature classifiers

Multiplicities/Arrays

Property values: configuration of data sets

Resource bindings: bindings proposal

In modes configurations: part of architecture design (extends/refined to)

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;

System configuration top.config extends top.design
( Sub1 => [10] , Sub2 => S.impl[15])
```

Array Sizes

V3 Complete Configuration Specification

- Connections internal to configuration
- Configuration parameter

```
System configuration top.configurable extends top.design
Prototypes
 Sub1 []; Sub2 [];

System implementation top.design
subcomponents
compute: system S[Size];
voter: system V;
Connections
C1: Port Sub1.pout -> Sub2.Pin { Connection_Pattern => (one_to_one);};

System configuration top.config extends top.design
( Sub1 => [10] , Sub2 => S.impl[15])
```

Array Sizes

V3 Complete Configuration Specification

- Same dimension with symbolic label

```
System configuration top.configurable
Prototypes
 SubSize: integer choices [constraint];
( Sub1 => [SubSize], Sub2.subsub2 => [SubSize]);
```

Array Sizes

V3 Partial Configuration Specification

- Configuration parameter

```
System top
Features
  Pin: in port;
  Pout: out port;
End top;
System configuration top.config extends top.design
Prototypes
Integer SubSize;
( Sub1 => [SubSize], Pout => [SubSize] );

System implementation top.design
subcomponents
Sub1 : system S;
Sub2 : system S;
Connections
C1: Port Sub1.pout -> Pout;
```

Variability Points

- Configuration of architecture structure
- Feature classifiers
- Array sizes
- Property values: configuration of data sets
- Resource bindings: bindings proposal
- In modes configurations: part of architecture design (extends/refined to)

Configurable properties

Parameterized configuration

- Explicit list of assignable properties

```
System configuration top.config extends top.design
Prototypes
  property Period;
```

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