Configuration of Variability Points

Peter Feiler
May 2016

Variability Points

- Configuration of architecture structure
  - Subcomponent type -> implementation
- Feature classifiers
  - Port data types
  - Access types
- Array sizes
- 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 prototype usage

```
Sub1.sub3: device sensorproto;
```

- As part of prototype actual assignment

```
Sub1.sub4: system gps(sensor => sensorproto);
```

Open and closed architectures

- Open architecture allows for reach down into subcomponents
- Closed architecture requires all variability points as prototypes
  - Parameterized system configuration

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 subsub;
End subsys.basic;

System 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;
```

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;
OR selections
Sub1 => Subsys.i;
Sub1.subsub1 => subsubsys.i;
```

Prototype actual syntax

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

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

System configuration top.fullconfig extends top.partconfig
subcomponents
Sub1.subsub1 : refined to system subsubsys.i;
```

Configurable Systems

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

System configuration PowerTrain.gas extends Powertrain.design
Prototypes
  engineselection: system gasengine;
End PowerTrain.gas;

System implementation Powertrain.design
Subcomponents
  myengine:  system gasengine;
End design;

System configuration car.config extends car.basic
subcomponents
 MyPowerTrain : refined to system PowerTrain.gas
    (engineSelection =>  gasengine.V4);
```

Match classifier name

Configurable Systems

```
System configuration PowerTrain.gas extends Powertrain.design
Prototypes
  engineselection: system gasengine applies to myengine;
End PowerTrain.gas;
```

Identify target

```
System configuration PowerTrain.gas extends Powertrain.design
Prototypes
  engineselection: system gasengine;
  myengine: refined to system engineselection;
End PowerTrain.gas;
```

Refine target

```
System implementation Powertrain.design
Subcomponents
  myengine:  system gasengine;
End design;
```

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
 speakerselection: system speakers;
End MySoundSystem.Selectablespeakers;

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

Nested Configurable Systems - 2

All variability points as top level prototypes

- Prototypes are mapped across multiple levels (speaker selection)

```
System configuration car.configurable
Prototypes
 engineselection: system gasengine;
 speakerselection: system speakers;

Subcomponents -- private
  PowerTrain:  system PowerTrain.gas (engineselection =>
engineselection);
  EntertainmentSystem:  system EntertainmentSystem.basic;
  EntertainmentSystem.soundsystem : refined to (speakerselection =>
speakerselection);
OR
EntertainmentSystem.soundsystem.speakerselection => speakerselection;
End car.configurable;

System configuration car.config extends car.configurable
( engineselection => engine.V4 ; speakerselection => 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 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 component type of the refinement must match the signature of the component type of the classifier being refined. Signature match is name mached subset of features with identical category and direction "Classifier_Match"-ed classifiers. [Potential expansion of features]

Configuration of Feature Classifiers

Ports take data classifiers

Features (and access features) take data, bus, subprogram (group) classifiers

V2 support

- Refined to: no classifier -> classifier, classifier substitution
- Prototype: category -> classifier, one level reference to prototype

V2 Support for Feature Classifier

```
System s
Prototypes
   bustype: bus;
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;
```

Configuration of Feature Classifiers (V3)

V3 support

- Matched name substitution
- Reach down configuration selection for subcomponent features/prototypes

V3 Support for Feature Classifiers

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

System configuration top.config
Prototypes
   datatype: data ;
subcomponents
Sub1 : system S;
Sub2 => system S;
```

Matching name in subcomponents

V3 Support for Feature Classifiers

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

System configuration top.config
Prototypes
   datatype: data ;
subcomponents
Sub1 => system S;
Sub1.p1 => datatype;
Sub2 => system S;
Sub2.p1 : refined to in data port datatype;
```

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)

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.

Array Sizes

V3 support

- Configuration parameter

```
System configuration top.design
Prototypes
 SubSize1: integer; SubSize2: integer;
subcomponents
Sub1 : system S [SubSize1];
Sub2 : system S [SubSize2];
```

- Property value

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

System configuration top.config extends top.design
Properties
Subsize => 10 applies to Sub1;
Subsize => 15 applies to Sub2;
```

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

Values for properties that have no assigned value

- Configuration properties section assigns values

Values for properties that already have values

- Scoped **constant**: Property can be assigned value only once
  - Cannot be configured
- Refinable property: type assigns default, implementation or sub can override
  - Configuration can override implementation
  - Configuration assignments cannot be changed

Parameterized configuration

- Explicit list of assignable properties (same as prototype)