Configuration of Variability Points

Peter Feiler
Jan 2016

Software Engineering Institute | Carnegie Mellon University

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 make selections for variability points of a chosen architecture design

- Finalize a selection

Architecture Design

Architecture design via extends and refines (V2)

- Addition of new and refinement of existing model elements
- In component types
  - Extend interface through addition of features, modes, flows, properties, modes
  - Refine interface features by identifying types on ports and access features
  - Override property values
- Component implementations
  - Extend implementation through addition of subcomponents, connections, flow implementations, modes, property values
  - Override property values including binding specifications
    - Own and contained component property values
  - Refine subcomponents by identifying implementation for type
    - Direct subcomponents only (V2)

Lower level refinement results in an up-proliferation of classifier extensions

Refinement Rules

For refined to (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]

Architecture Design

Architecture design via extends and prototypes (V2)

- Substitution of actual selection for classifier variability point
  - Prototype specification: category and classifier constrained
    - Usage by reference to prototype ID
    - Multiple uses of same classifier (constraint)
  - Prototype actual assignment
    - As part of extends, e.g., A extends B (proto1 => system X)
    - As part of subcomponent declaration or refined to, e.g., sub: refined to system B (proto1 => system X)
      - Actual can be prototype ID of enclosing component
- In component types
  - Choice of classifier on ports and access points
  - Substitution rules see next slide
- Component implementations
  - Choice of classifier for direct subcomponents
  - Substitution rules see next slide

Lower level prototype results in an up-accumulation of prototypes

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]

V3 Proposal

Ability to make selection multiple levels down in architecture design

- As part of refined to

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

- As part of prototype actual assignment

Separate construct to specify configurations

- Clear indication that we configure an architecture rather than expand the design space
- We can only complete selections for variability points, not change a previously made selection
- Can be referenced in lieu of a component type or implementation

```
System top.config1 configures top.basic
selections
Sub1.sub11 => system Component.Implementationy ;
```

Uses prototype actual like syntax.
We could use *refined to* syntax

Refinement of Architecture Design

Ability to refine across multiple architecture levels

- Reduces need for classifier extensions of intermediate levels
- Selections may be configurable themselves

```
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;
  Sub1.subsub1 : refined to system subsubsys.i;
```

Configuration of an Open Architecture

We are responsible for several levels of the component hierarchy

- We have visibility into the realization of component implementations
- We configure subcomponents without identified implementation
  - No replacement of a present implementation
- Selections may be configurable themselves

```
System top.config1 configures top.basic
selections
Sub1 => system Subsys.i;
Sub1.subsub1 => system subsubsys.i;
OR
Sub1 => system Subsys.i (subsub1 => system subsubsys.i);
```

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

```
System top.partconfig configures top.basic
selections
Sub1 => system Subsys.i;

System top.fullconfig configures partconfig
selections
Sub1.subsub1 => system subsubsys.i;
```

Configuration of an Partially Closed Architecture

We have no visibility into the realization of some subsystems

- A closed subsystem can be at any level in the hierarchy
- Variability points must be specified as part of the visible specification
  - All classifier variability points are expressed as prototype

Configuration of an Partially Closed Architecture

Powertrain is a closed subsystem

```
System implementation car.basic
Subcomponents
  PowerTrain:  system PowerTrain;
  EntertainmentSystem:  system EntertainmentSystem.basic;
End car.basic;
System variant PowerTrain.gas for Powertrain.design
Prototypes  -- public
  engineselection: system gasengine;
System implementation Powertrain.design
Subcomponents  -- private
  myengine:  system engineselection;
End PowerTrain.gas;

System  car.config configures car.basic
Selections
 PowerTrain => PowerTrain.gas ( engineselection => gasengine.V4 );
OR
 PowerTrain => PowerTrain.gas;
 PowerTrain.engineselection => gasengine.V4;
```

Nested Closed Subsystems

Sound system inside the entertainment system is closed

- Speaker selection as variability point

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

System implementation MySoundSystem.Selectablespeakers
Prototypes -- public
 speakerselection: system speakers;
Subcomponents -- private
  amplifier:  system Amplifier.Kenwood;
  speakers: system speakers;
End MySoundSystem.Selectablespeakers;
```

Nested Closed Subsystems

All variability points as top level prototypes

- Prototypes are mapped across multiple levels (speaker selection)

```
System implementation car.configurable
Prototypes  -- public
 engineselection: system gasengine;
 speakerselection: system speakers;
Subcomponents -- private
  PowerTrain:  system PowerTrain.gas;
  EntertainmentSystem:  system EntertainmentSystem.basic;
  EntertainmentSystem.soundsystem.speakerselection : refined to system
speakerselection;
OR
 EntertainmentSystem:  system EntertainmentSystem.basic
          (soundsystem.speakerselection => speakerselection);
End car.configurable;

System  car.config configures car.configurable
Selections
 engineselection => engine.V4 ;
 speakerselection => Speakers.Bose;
```

Making car closed in the design space.
All variability points as prototypes.

Note: V2 allows a mix of variability point via prototype and open.

Partial Configuration

Remaining variability points

- Remaining unbound prototypes (V2 prototype rules)
- For open architecture: subcomponent classifier
  - Refined to can replace implementations (V2)
  - Configuration selection cannot be overridden (V3)

```
System  car.V4config configures car.configurable
Selections
 engineselection => engine.V4 ;

System  car.myconfig configures car.v4config
Selections
 speakerselection => Speakers.Bose;
```

Prototypes for Configurations

Partial configuration that users can fully configure

- Making a design closed as configuration declaration
- Remaining variability points a prototypes
- Selection partially exposes hierarchy

```
System car.configurable configures car.basic
Prototypes  -- public
 engineselection: system gasengine;
 speakerselection: system speakers;
selections -- private
  PowerTrain => system PowerTrain.gas;
  EntertainmentSystem.soundsystem.speakerselection => system
speakerselection;
End car.configurable;

System  car.config configures car.configurable
Selections
 engineselection => engine.V4 ;
 speakerselection => Speakers.Bose;
```

Should We Allow Reconfiguration?

What is the purpose of being able to change an implementation?

- Alexey: value in allowing change to a preconfigured system, e.g., a box in which some boards in some slots can be changed.
- Default implementation (e.g., Simulink configurable subsystem block)

In architecture design

- Current substitution rules allow implementation replacement
  - Result is different set of subcomponents
  - Refined to of subcomponents with implementations
  - If an implementation was declared in prototype declarations
    - But no overriding of prototype actuals
- Replacement by type extension or signature match
  - may result additional features requiring connection declarations

In configuration

- Selections cannot be changed (V3)

Reconfiguration Options

Option 1: Take different point of view

- We are specifying a delta/diff
- We should view both implementation configurations as configured/refined from a common architecture (type only)

Option 2: ok in design, not for configurations

- Supported by current rules (mostly)
- Need to allow prototype actual override

Option 3: support concept of default implementation

- Explicit indicator if replacement is allowed (**default**)
- Explicit indicator if replacement is not allowed (**final**)

Configuration of Feature Classifiers

Ports take data classifiers

Access features take data, bus, subprogram (group) classifiers

Objective:

- Supply data type/bus type

V2 support

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

V3 support

- As for subcomponent configuration
- Reach down configuration selection for subcomponent features/prototypes

V2 Support for Feature Classifiers

```
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;
```

V2 Support for Feature Classifiers

```
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.design
Subcomponents
 Sub1:  system S (bustype => bus Ethernet);
 Sub2:  system SRefined.basic;
End subsys.design;
```

Configuration of Feature Classifiers (V3)

V3 support

- As for subcomponent configuration
- Reach down configuration selection for subcomponent features/prototypes

Discussion

- Replacement of existing classifier
- Scoped multiple level configuration/replacement
  - Within component type, package, component hierarchy
  - Along connections

V3 Support for Feature Classifiers

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

System implementation top.design
Subcomponents
 Sub1:  refined to system S (bustype => bus Ethernet);
 Sub1.p1 : refined to data DT;
 Sub2:  system S.basic (bustype => bus Ethernet);
End subsys.design;

System top.config configures top.basic
Selections
Sub1 => system S.basic;
Sub1.p1 => data DT;
  Sub2 => system Srefined.basic;
```

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

Configuration Specification Proposal

- Additional uses of **refined to**
  - Add or replace property values with inline property associations {}
    - the properties subclause lets us specify sets (configurations) of property values (recursively) for subcomponents and other model elements through contained property associations
  - Add dimension size to arrays (no replacement)
  - Add or replace in modes configuration to existing (direct) subcomponents
    - In modes only in subcomponent and in configuration section?
    - Do we want to have replacement?
    - Do we want to be able to specify multiple in modes configurations via refinement?
    - Option 1: in modes only in original subcomponent declaration
    - Option 2: in modes only in configuration section

Improvement Candidates

Need for refined to of prototypes?

- Restrict to a subtype
  - Implementation extension introduces own prototypes
  - Implementation extension may assign actuals to prototypes
- Restrict to implementation, which in the design can be changed

Reduce wordiness of syntax

- Do we require the category when specifying the prototype actual or refined to?
- Do we requirement category indicator for connections?