Towards Configuration Language for Universal Variability Language

H. **Sayyid** Fadhlillah
Christian Doppler Lab VaSiCS
LIT | Cyber-Physical Systems Lab
Johannes Kepler University Linz

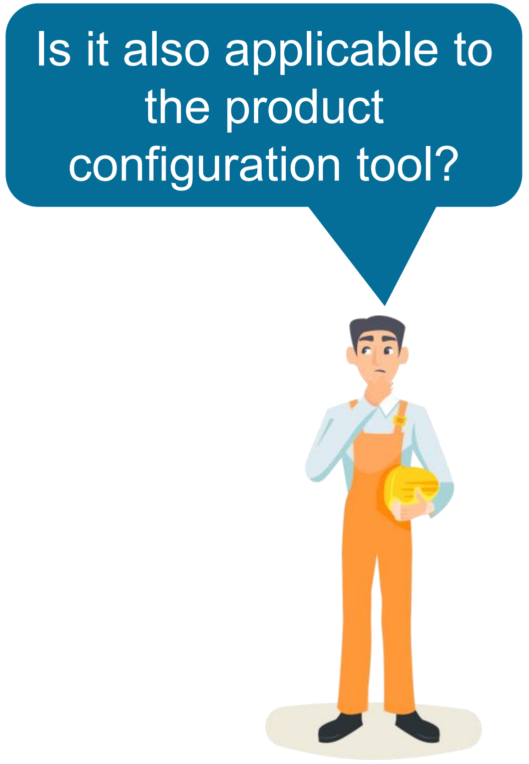
Motivation

```
namespace "Ice Cream"

features
    "Ice Cream" {extended__ true}
        mandatory
            Category
                alternative
                    Popsicle {Price 1}
                    Scoop {Price 2}
                    Waffle {Price 0.7}
            Flavors
                or
                    Lemon
                    Chocolate
                        alternative
                            White
                            Dark
            Container
                alternative
                    Stick
                    Cup
                    Cone
        optional
            "Name of customer" {feature_type 'String'}

constraints
                        Popsicle => Stick
                        Scoop => Cup | Cone
```

Community effort towards unified language

One interchangeable variability language format for different tools

Currently represented using textual format



Existing UVL Configuration Tool

```
namespace "Ice Cream"

features
   "Ice Cream" {extended__ true}
      mandatory
         Category
            alternative
               Popsicle {Price 1}
               Scoop {Price 2}
               Waffle {Price 0.7}
         Flavors
            or
               Lemon
               Chocolate
                  alternative
                     White
                     Dark
         Container
            alternative
               Stick
               Cup
               Cone
      optional
         "Name of customer" {feature_type 'String'}

constraints
                  Popsicle => Stick
                  Scoop => Cup | Cone
```

Feature Configuration

Rabiser et. al., Requirements for product derivation support: Results from a systematic literature review and an expert survey, Information and Software Technology, Volume 52, Issue 3, 2010, Pages 324-346, ISSN 0950-5849.

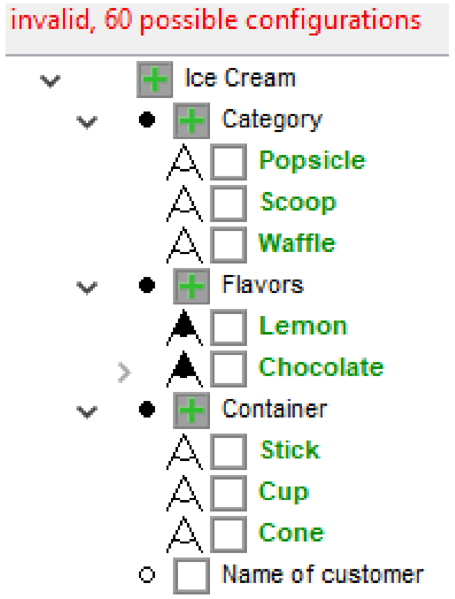
Existing UVL Configuration Tool

Automated and interactive variability resolution

End-user guidance

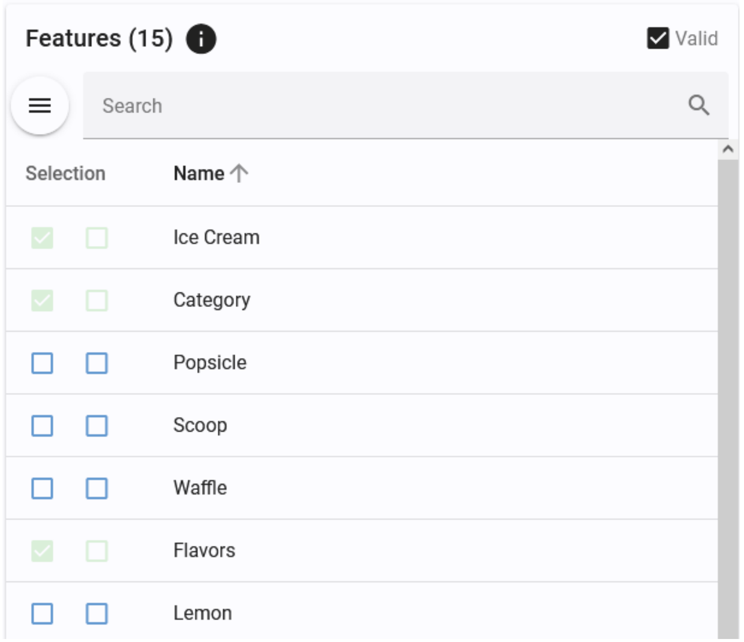
Adaptability and extension

Flexible and user-specific visualizations of variability


invalid, 60 possible configurations

- Ice Cream
  - Category
    - Popsicle
    - Scoop
    - Waffle
  - Flavors
    - Lemon
    - Chocolate
  - Container
    - Stick
    - Cup
    - Cone
  - Name of customer

Variability.dev


Features (15) ✓ Valid

Search

Selection	Name
✓	Ice Cream
✓	Category
	Popsicle
	Scoop
	Waffle
✓	Flavors
	Lemon

Rabiser et. al., Requirements for product derivation support: Results from a systematic literature review and an expert survey, Information and Software Technology, Volume 52, Issue 3, 2010, Pages 324-346, ISSN 0950-5849.

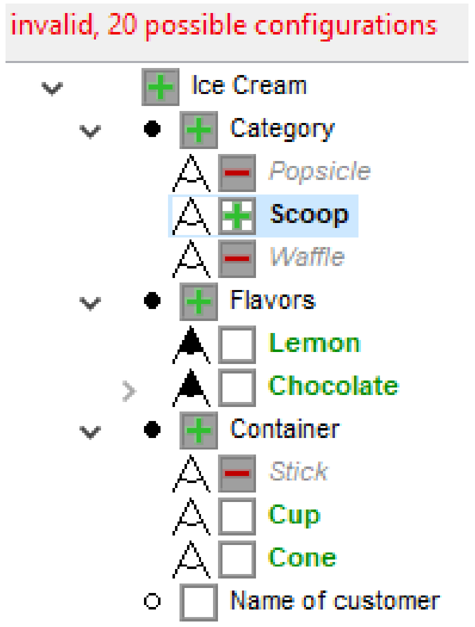
Existing UVL Configuration Tool

Automated and interactive variability resolution

End-user guidance

Adaptability and extension

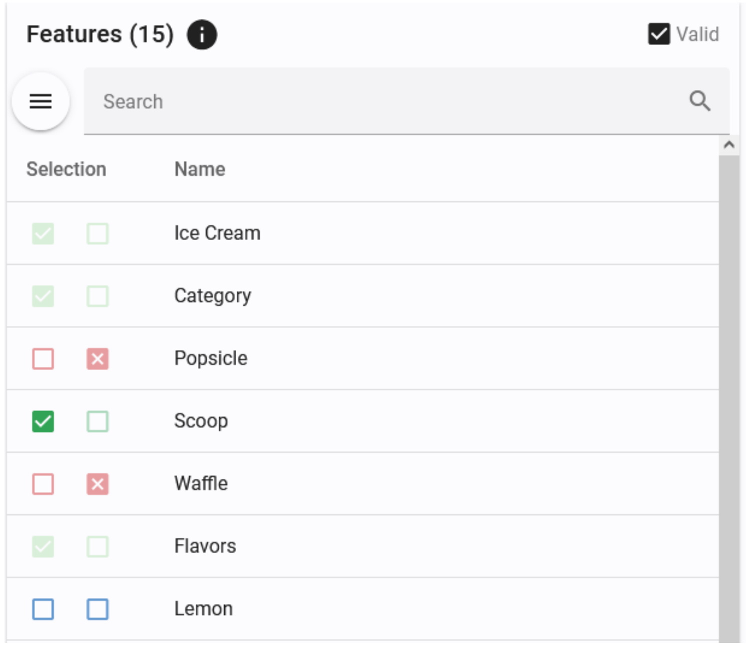
Flexible and user-specific visualizations of variability



invalid, 20 possible configurations

- Ice Cream
  - Category
    - Popsicle
    - Scoop
    - Waffle
  - Flavors
    - Lemon
    - Chocolate
  - Container
    - Stick
    - Cup
    - Cone
  - Name of customer

Variability.dev



Features (15) ✓ Valid

Search

Selection		Name
✓	☐	Ice Cream
✓	☐	Category
☐	☒	Popsicle
✓	☐	Scoop
☐	☒	Waffle
✓	☐	Flavors
☐	☐	Lemon

Existing UVL Configuration Tool

FeatureIDE

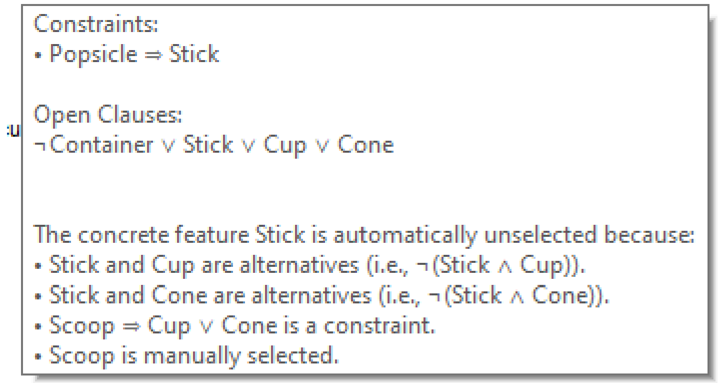
Variability.dev

Automated and interactive variability resolution

End-user guidance

Adaptability and extension

Flexible and user-specific visualizations of variability

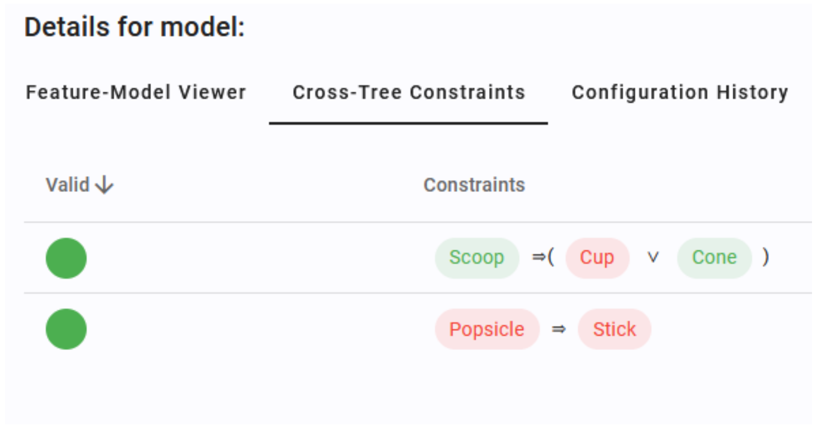


:u

Constraints:
- Popsicle ⇒ Stick

Open Clauses:
¬Container ∨ Stick ∨ Cup ∨ Cone

The concrete feature Stick is automatically unselected because:
- Stick and Cup are alternatives (i.e., ¬(Stick ∧ Cup)).
- Stick and Cone are alternatives (i.e., ¬(Stick ∧ Cone)).
- Scoop ⇒ Cup ∨ Cone is a constraint.
- Scoop is manually selected.



Details for model:

Feature-Model Viewer | Cross-Tree Constraints | Configuration History

Valid ↓	Constraints
●	Scoop ⇒(Cup ∨ Cone)
●	Popsicle ⇒ Stick

Existing UVL Configuration Tool

IDE
Feature

Variability.dev

Automated and interactive variability resolution

End-user guidance

Adaptability and extension

Flexible and user-specific visualizations of variability

"… adaptable and extensible to support different organization and technological contexts"

Existing UVL Configuration Tool

FeatureIDE

Variability.dev

Automated and interactive variability resolution

Available on GitHub

End-user guidance

Adaptability and extension

API for FeatureIDE Library

Flexible and user-specific visualizations of variability

Existing UVL Configuration Tool

FeatureIDE

Variability.dev

Automated and interactive variability resolution

End-user guidance

Adaptability and extension

Flexible and user-specific visualizations of variability

"Rich graphic representations while also customizable depending on the user contexts (e.g., role or tasks)"

Existing UVL Configuration Tool

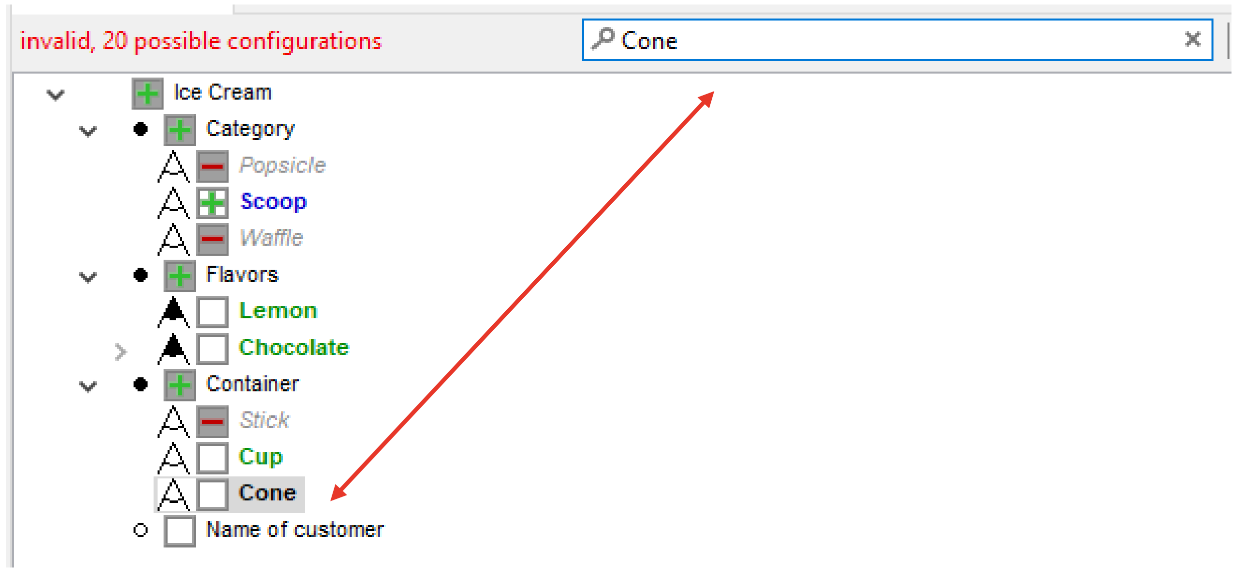
Variability.dev

Automated and interactive variability resolution

End-user guidance

Adaptability and extension

Flexible and user-specific visualizations of variability


invalid, 20 possible configurations

Cone

- Ice Cream
  - Category
    - Popsicle
    - Scoop
    - Waffle
  - Flavors
    - Lemon
    - Chocolate
  - Container
    - Stick
    - Cup
    - Cone
  - Name of customer

Existing UVL Configuration Tool

Automated and interactive variability resolution

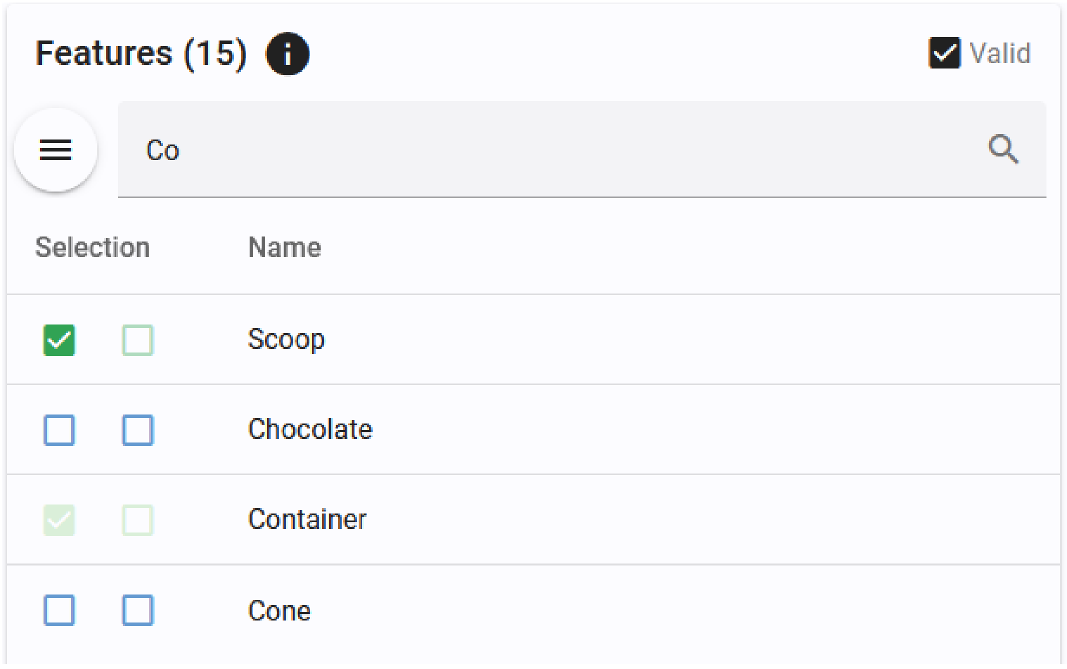
End-user guidance

Adaptability and extension

Flexible and user-specific visualizations of variability

FeatureIDE

Variability.dev


Features (15)

☑ Valid

Co

Selection		Name
✓	☐	Scoop
☐	☐	Chocolate
✓	☐	Container
☐	☐	Cone

Existing UVL Configuration Tool

FeatureIDE

Variability.dev

Automated and interactive variability resolution

End-user guidance

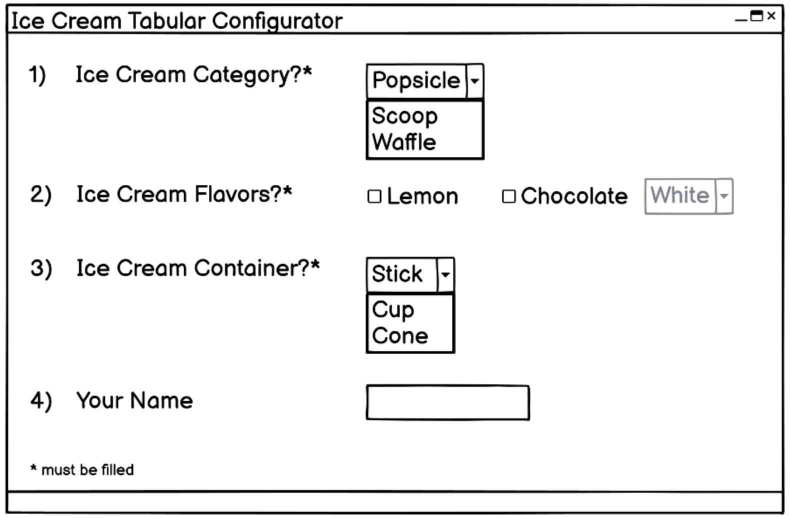
Adaptability and extension

Flexible and user-specific visualizations of variability

“Flexible Visualization” can be interpreted into several thing

Flexible Configuration: Custom Representation

```
namespace "Ice Cream"

features
  "Ice Cream" {extended__ true}
    mandatory
      Category
        alternative
          Popsicle {Price 1}
          Scoop {Price 2}
          Waffle {Price 0.7}
      Flavors
        or
          Lemon
          Chocolate
            alternative
              White
              Dark
      Container
        alternative
          Stick
          Cup
          Cone
    optional
      "Name of customer" {feature_type 'String'}

constraints
          Popsicle => Stick
          Scoop => Cup | Cone
```



Ice Cream Tabular Configurator

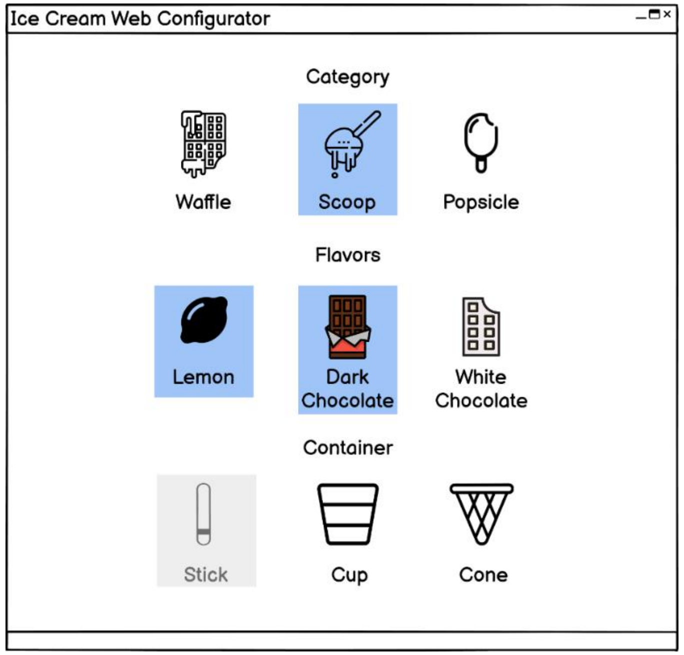
1) Ice Cream Category?* Popsicle / Scoop / Waffle

2) Ice Cream Flavors?* ☐ Lemon ☐ Chocolate White

3) Ice Cream Container?* Stick / Cup / Cone

4) Your Name

* must be filled

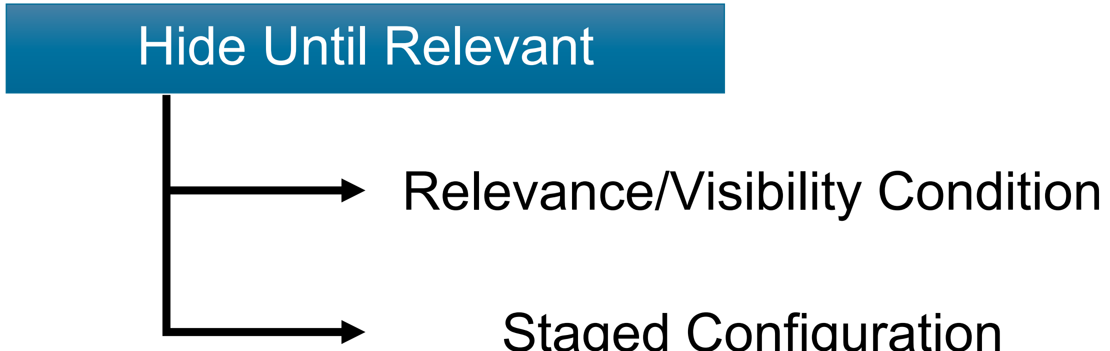


Ice Cream Web Configurator

Flexible Configuration: Sequences

```
namespace "Ice Cream"

features
    "Ice Cream" {extended__ true}
        mandatory
            Category
                alternative
                    Popsicle {Price 1}
                    Scoop {Price 2}
                    Waffle {Price 0.7}
            Flavors
                or
                    Lemon
                    Chocolate
                        alternative
                            White
                            Dark
            Container
                alternative
                    Stick
                    Cup
                    Cone
        optional
            "Name of customer" {feature_type 'String'}

constraints
                Popsicle => Stick
                Scoop => Cup | Cone
```

Only relevant if category is not **Popsicle**



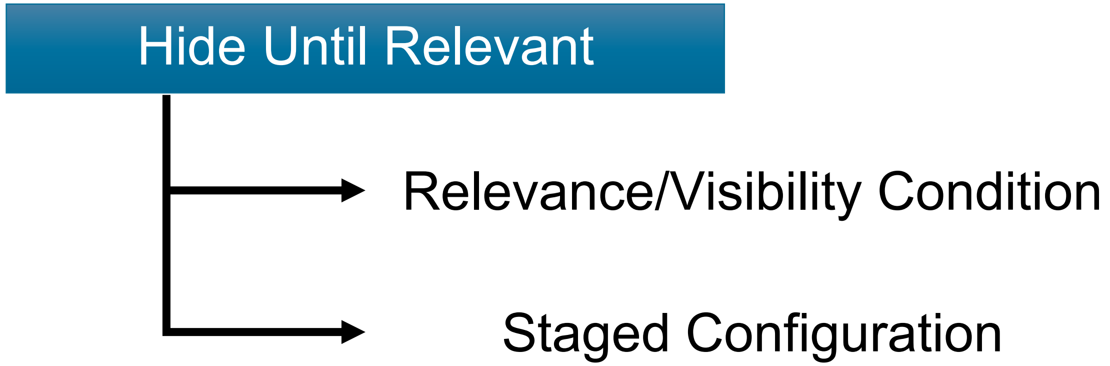
"The relevancy of a decision may depend on other decisions"

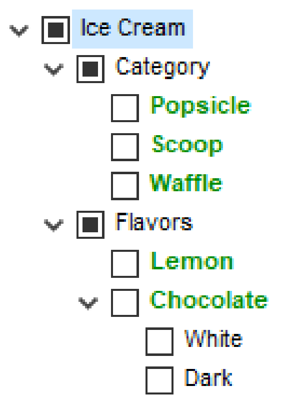
Schmid and John, A customizable approach to full lifecycle variability management, Science of Computer Programming, Volume 53, Issue 3, 2004, Pages 259-284, ISSN 0167-6423.

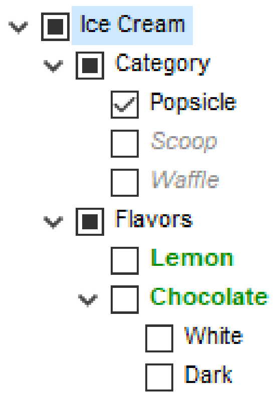
Flexible Configuration: Sequences

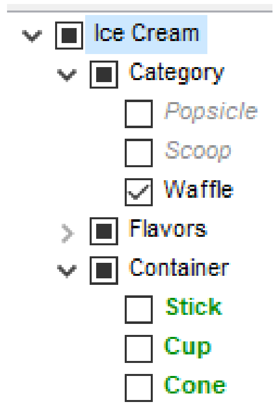
```
namespace "Ice Cream"

features
   "Ice Cream" {extended__ true}
      mandatory
         Category
            alternative
               Popsicle {Price 1}
               Scoop {Price 2}
               Waffle {Price 0.7}
         Flavors
            or
               Lemon
               Chocolate
                  alternative
                     White
                     Dark
         Container
            alternative
               Stick
               Cup
               Cone
      optional
         "Name of customer" {feature_type 'String'}

constraints
            Popsicle => Stick
            Scoop => Cup | Cone
```

Only relevant if category is not **Popsicle**



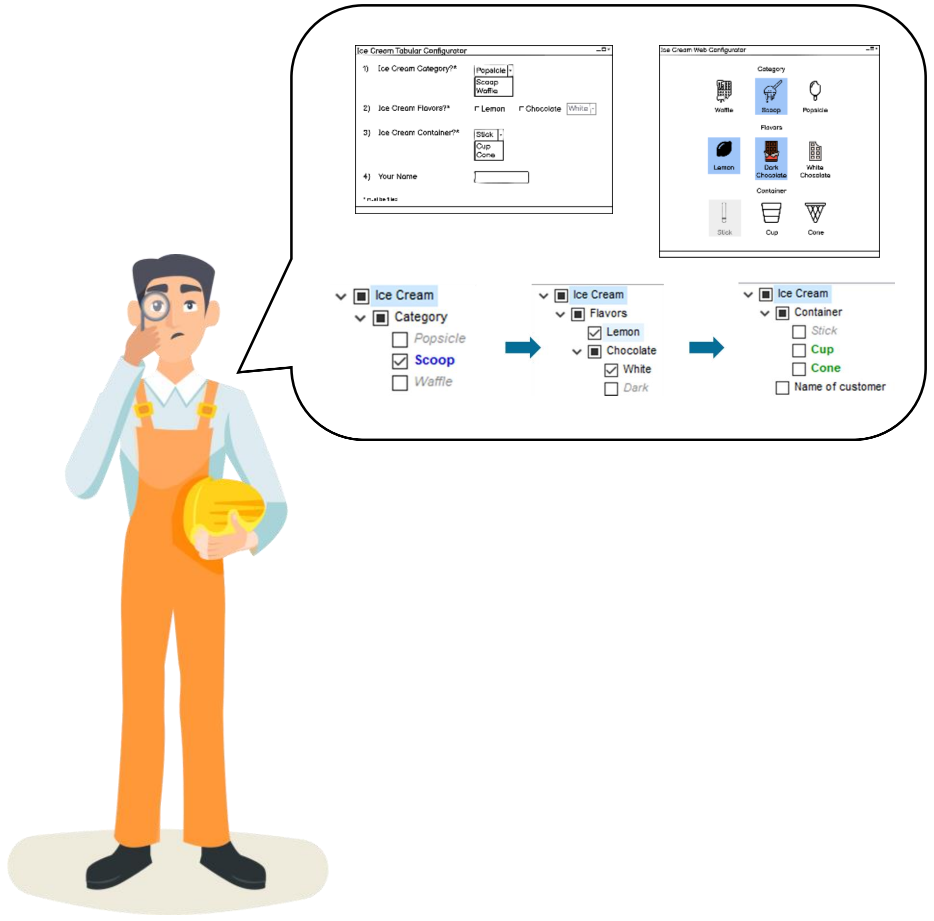






Schmid and John, A customizable approach to full lifecycle variability management, Science of Computer Programming, Volume 53, Issue 3, 2004, Pages 259-284, ISSN 0167-6423.

Customizable Configuration Aspects



User Interface Elements

Can be customized into **any graphical representation** depending on the context

Configuration Sequences

Can be similar regardless of the graphical representation

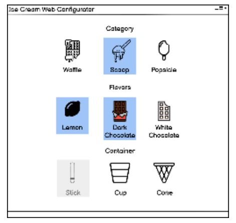
Proposal for UVL Configuration Language

A variability model **can be configured in different ways**

Proposal for UVL Configuration Language

```
namespace "Ice Cream"

features
    "Ice Cream" {extended__ true}
        mandatory
            Category
                alternative
                    Popsicle {Price 1}
                    Scoop {Price 2}
                    Waffle {Price 0.7}
            Flavors
                or
                    Lemon
                    Chocolate
                        alternative
                            White
                            Dark
            Container
                alternative
                    Stick
                    Cup
                    Cone
        optional
            "Name of customer" {feature_type 'String'}

constraints
            Popsicle => Stick
            Scoop => Cup | Cone
```

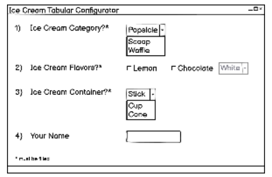
Configuration Type 1


Ice Cream Web Configurator

Category: Waffle, Scoop, Popsicle

Flavors: Lemon, Dark Chocolate, White Chocolate

Container: Stick, Cup, Cone


Ice Cream Tabular Configurator

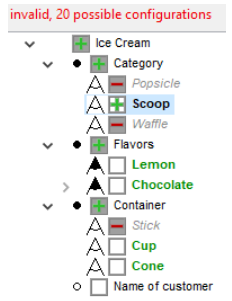
1) Ice Cream Category?* Popsicle / Scoop / Waffle

2) Ice Cream Flavors?* Lemon, Chocolate, White

3) Ice Cream Container?* Stick / Cup / Cone

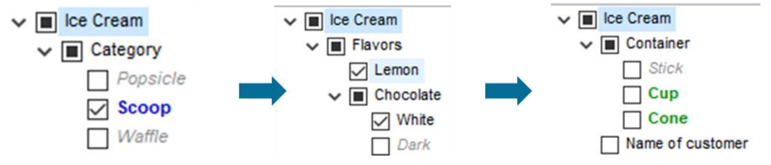
4) Your Name

* must be filled


invalid, 20 possible configurations

- Ice Cream
  - Category
    - Popsicle
    - Scoop
    - Waffle
  - Flavors
    - Lemon
    - Chocolate
  - Container
    - Stick
    - Cup
    - Cone
  - Name of customer

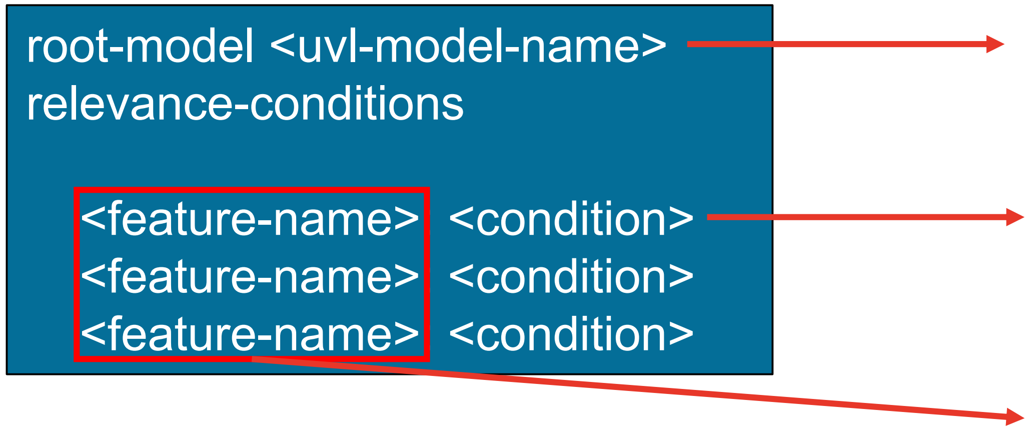
Configuration Type 2


- ■ Ice Cream
  - ■ Category
    - ☐ Popsicle
    - ☑ Scoop
    - ☐ Waffle

- ■ Ice Cream
  - ■ Flavors
    - ☑ Lemon
    - ■ Chocolate
      - ☑ White
      - ☐ Dark

- ■ Ice Cream
  - ■ Container
    - ☐ Stick
    - ☐ Cup
    - ☐ Cone
  - ☐ Name of customer

Proposal for UVL Configuration Language



```
root-model <uvl-model-name>
relevance-conditions

    <feature-name>  <condition>
    <feature-name>  <condition>
    <feature-name>  <condition>
```

Model to be configured

Can be written similar to a feature constraint

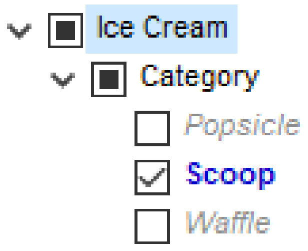
Order of the condition can be interpreted as **configuration sequences**

Configuration Language Example: Single Product Line

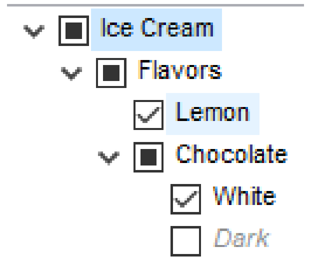
```
namespace "Ice Cream"

features
  "Ice Cream" {extended__ true}
    mandatory
      Category
        alternative
          Popsicle {Price 1}
          Scoop {Price 2}
          Waffle {Price 0.7}
      Flavors
        or
          Lemon
          Chocolate
            alternative
              White
              Dark
      Container
        alternative
          Stick
          Cup
          Cone
    optional
      "Name of customer" {feature_type 'String'}

constraints
            Popsicle => Stick
            Scoop => Cup | Cone
```

Shown first since there is no relevance conditions

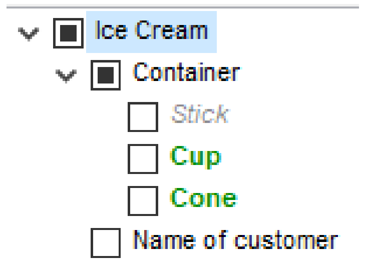
```
root-model ice-cream.uvl
relevance-conditions

    Flavors: Popsicle | Scoop | Waffle
    Container: !Popsicle
```



- ▣ Ice Cream
  - ▣ Category
    - ☐ Popsicle
    - ✓ Scoop
    - ☐ Waffle



- ▣ Ice Cream
  - ▣ Flavors
    - ✓ Lemon
    - ▣ Chocolate
      - ✓ White
      - ☐ Dark



- ▣ Ice Cream
  - ▣ Container
    - ☐ Stick
    - ☐ Cup
    - ☐ Cone
  - ☐ Name of customer

Configuration Language Example: Multi Product Line

```
namespace SimplifyBerkeleyDB
imports
    nio_feature as nio
    io_feature as io
    logging as logging
    persistency as persistencyFeatures

features
  SimplifyBerkeleyDB {abstract true}
    mandatory
      Persistency {abstract true}
        mandatory
          FIOFeature
            alternative
              io.IOFeature
              nio.NIOFeature
        optional
          persistencyFeatures.Persistency
    optional
      logging.Logging
```

```
namespace Logging

features
  Logging {abstract true}
    mandatory
      Base
    optional
      DBLog
      Console
      File
```

```
namespace Persistency

features
  Persistency {abstract true}
    or
      CheckPointer
        optional
          Daemon
          Bytes
          Time
      EnvironmentLock
      FileHandleCache
      Checksum
```

```
namespace IOFeature

features
  IOFeature {abstract true}
    mandatory
      BaseIO
    optional
      Synchronized
```

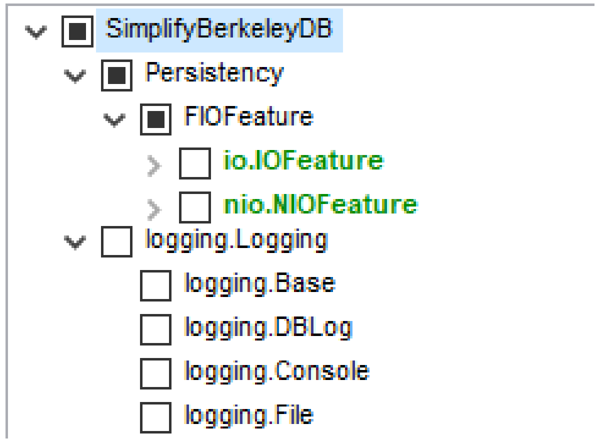
```
namespace NIOFeature

features
  NIOFeature {abstract true}
    mandatory
      NIOType
        optional
          Chunked
          Pure
    optional
      DirectNIO
```

```
root-model simplify-berkeley.uvl
relevance-conditions

 logging.Logging: io.IOFeature ||
nio.NIOFeature

 persistencyFeatures.Persistency:
logging.Logging

 persistencyFeatures.CheckPointer:
nio.DirectNIO
```

Configuration Language Example: Multi Product Line

```
namespace SimplifyBerkeleyDB
imports
   nio_feature as nio
   io_feature as io
   logging as logging
   persistency as persistencyFeatures

features
  SimplifyBerkeleyDB {abstract true}
   mandatory
     Persistency {abstract true}
       mandatory
         FIOFeature
           alternative
             io.IOFeature
             nio.NIOFeature
       optional
         persistencyFeatures.Persistency
   optional
     logging.Logging
```

```
root-model simplify-berkeley.uvl
relevance-conditions

 persistencyFeatures.Persistency:
logging.Logging

 persistencyFeatures.CheckPointer:
nio.DirectNIO
```



```
▣ SimplifyBerkeleyDB
  ▣ Persistency
    ▣ FIOFeature
      ☐ io.IOFeature
      ☐ nio.NIOFeature
  ☐ logging.Logging
    ☐ logging.Base
    ☐ logging.DBLog
    ☐ logging.Console
    ☐ logging.File
```

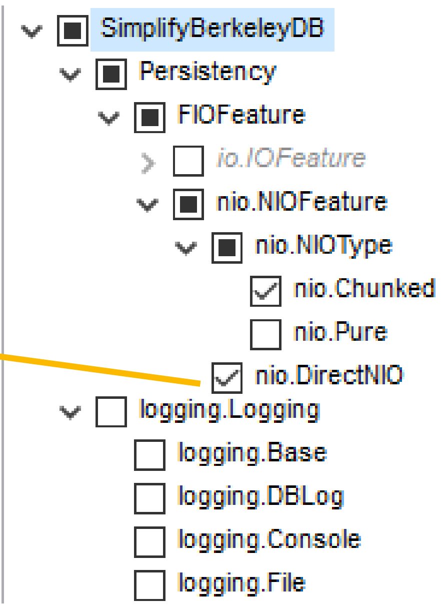
Initial State

Configuration Language Example: Multi Product Line

```
namespace SimplifyBerkeleyDB
imports
   nio_feature as nio
   io_feature as io
   logging as logging
   persistency as persistencyFeatures

features
  SimplifyBerkeleyDB {abstract true}
   mandatory
     Persistency {abstract true}
       mandatory
         FIOFeature
           alternative
             io.IOFeature
             nio.NIOFeature
       optional
         persistencyFeatures.Persistency
   optional
     logging.Logging
```

```
root-model simplify-berkeley.uvl
relevance-conditions

 persistencyFeatures.Persistency:
logging.Logging

 persistencyFeatures.CheckPointer:
nio.DirectNIO
```

1. Out of Order
2. Unresolved hierarchy visibility



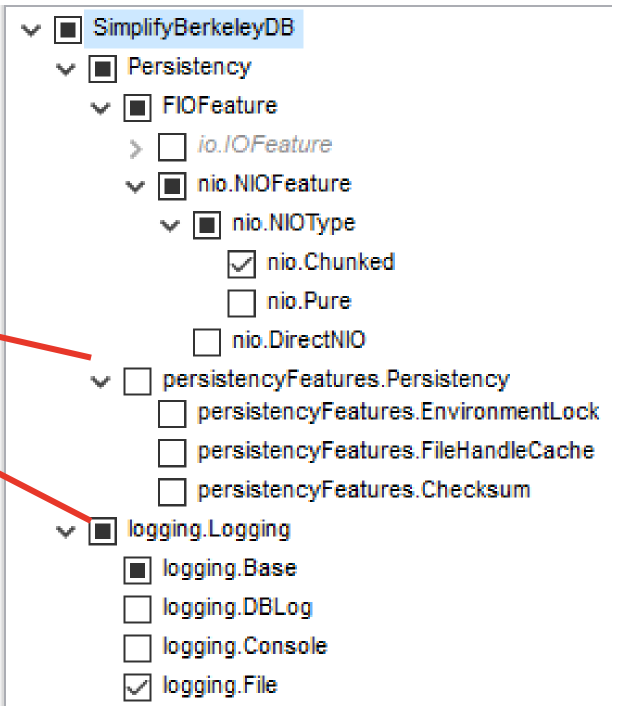
```
⌄ ■ SimplifyBerkeleyDB
  ⌄ ■ Persistency
    ⌄ ■ FIOFeature
      › □ io.IOFeature
      ⌄ ■ nio.NIOFeature
        ⌄ ■ nio.NIOType
            ✓ nio.Chunked
            □ nio.Pure
          ✓ nio.DirectNIO
  ⌄ □ logging.Logging
      □ logging.Base
      □ logging.DBLog
      □ logging.Console
      □ logging.File
```

Selecting DirectNIO

Configuration Language Example: Multi Product Line

```
namespace SimplifyBerkeleyDB
imports
    nio_feature as nio
    io_feature as io
    logging as logging
    persistency as persistencyFeatures

features
  SimplifyBerkeleyDB {abstract true}
    mandatory
      Persistency {abstract true}
        mandatory
          FIOFeature
            alternative
              io.IOFeature
              nio.NIOFeature
        optional
          persistencyFeatures.Persistency
    optional
      logging.Logging
```

```
root-model simplify-berkeley.uvl
relevance-conditions

 persistencyFeatures.Persistency:
logging.Logging

 persistencyFeatures.CheckPointer:
nio.DirectNIO
```



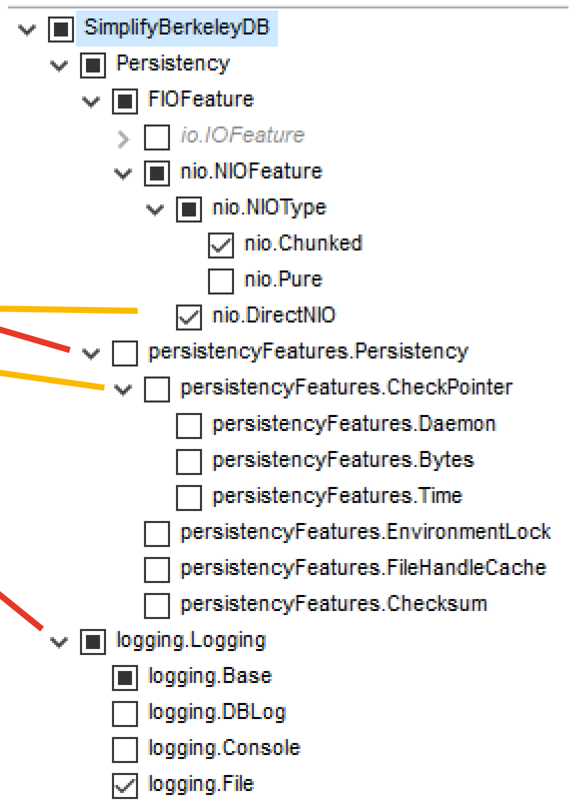
- ▣ SimplifyBerkeleyDB
  - ▣ Persistency
    - ▣ FIOFeature
      - ☐ io.IOFeature
      - ▣ nio.NIOFeature
        - ▣ nio.NIOType
          - ☑ nio.Chunked
          - ☐ nio.Pure
        - ☐ nio.DirectNIO
    - ☐ persistencyFeatures.Persistency
      - ☐ persistencyFeatures.EnvironmentLock
      - ☐ persistencyFeatures.FileHandleCache
      - ☐ persistencyFeatures.Checksum
  - ▣ logging.Logging
    - ▣ logging.Base
    - ☐ logging.DBLog
    - ☐ logging.Console
    - ☑ logging.File

Selecting Logging

Configuration Language Example: Multi Product Line

```
namespace SimplifyBerkeleyDB
imports
  nio_feature as nio
  io_feature as io
  logging as logging
  persistency as persistencyFeatures

features
 SimplifyBerkeleyDB {abstract true}
  mandatory
   Persistency {abstract true}
    mandatory
     FIOFeature
      alternative
       io.IOFeature
       nio.NIOFeature
    optional
     persistencyFeatures.Persistency
  optional
   logging.Logging
```

```
root-model simplify-berkeley.uvl
relevance-conditions

 persistencyFeatures.Persistency:
logging.Logging

 persistencyFeatures.CheckPointer:
nio.DirectNIO
```



```
⌄ ■ SimplifyBerkeleyDB
  ⌄ ■ Persistency
    ⌄ ■ FIOFeature
      › □ io.IOFeature
      ⌄ ■ nio.NIOFeature
        ⌄ ■ nio.NIOType
            ✓ nio.Chunked
            □ nio.Pure
          ✓ nio.DirectNIO
    ⌄ □ persistencyFeatures.Persistency
      ⌄ □ persistencyFeatures.CheckPointer
          □ persistencyFeatures.Daemon
          □ persistencyFeatures.Bytes
          □ persistencyFeatures.Time
        □ persistencyFeatures.EnvironmentLock
        □ persistencyFeatures.FileHandleCache
        □ persistencyFeatures.Checksum
  ⌄ ■ logging.Logging
      ■ logging.Base
      □ logging.DBLog
      □ logging.Console
      ✓ logging.File
```

Possible Challenges

```
namespace "Ice Cream"

features
    "Ice Cream" {extended__ true}
        mandatory
            Category
                alternative
                    Popsicle {Price 1}
                    Scoop {Price 2}
                    Waffle {Price 0.7}
            Flavors
                or
                    Lemon
                    Chocolate
                        alternative
                            White
                            Dark
            Container
                alternative
                    Stick
                    Cup
                    Cone
        optional
            "Name of customer" {feature_type 'String'}

constraints
                    Popsicle => Stick
                    Scoop => Cup | Cone
```

```
root-model ice-cream.uvl
relevance-conditions

    Flavors: Popsicle | Scoop | Waffle
    Container: Popsicle
```

Conflict between relevance conditions and feature constraints

- Reducing configuration space
- Unsolvable product configuration

Formal description of relevance conditions

- Reducing ambiguity for tool support implementation

Your Feedback are Meaningful

Additional requirements for configuration language?
Possible Collaboration for Tool Implementation?

JKU
LINZ INSTITUTE
OF TECHNOLOGY

Thank you!

Hafiyyan **Sayyid** Fadhlillah | hafiyyan.fadhlillah@jku.at
Christian Doppler Lab VaSiCS
LIT | Cyber-Physical Systems Lab
Johannes Kepler University Linz

PRIMETALS
TECHNOLOGIES

ober
österreich

Federal Ministry
Republic of Austria
Digital and
Economic Affairs