Integrated Management of Variability in Space and Time in Software Families
Using Hyper Feature Models and Delta Modeling

Christoph Seidl
Ina Schaefer
Uwe Aßmann
### Background

- **Variability in Space**
  - Configuration
  - **Variant** of system

- **Variability in Time**
  - Evolution
  - **Version** of system

- **Variability Realization Mechanisms (Excerpt)**

**Desired Variant**

- Software Variant

**Subtractive Variability**

- 150% Model

**Delta Modeling**

- Software Variant A
Cross-Tree Constraints

1. Autonomous → Detection
2. Keyboard ∨ Gamepad → Webservice
Development History of TurtleBot Driver

Lessons Learned
- Many updates!
- Old versions are still out there
- Incompatibilities of versions!

Resulting Problems
- Cannot model the evolution on conceptual level
- Cannot derive "older" variants!
Requirements for Deriving "Older" Variants

1. Variability Model that can handle Variability in Time
2. Variability Realization Mechanism that can handle Variability in Time
3. Variant Derivation Process that can handle Variability in Time
Solution Overview

Variability Model
- Feature Models

Variability Realization Mechanism
- Delta Modeling

Variant Derivation Process
- Transformation

1. Extension for Variability in Time
2. Extension for Variability in Time
3. Extension for Variability in Time
Feature Models: No Variability in Time

- Constraints caused by evolution (excerpt)
  - iClebo Kobuki is incompatible with TurtleBot v1.0
  - iClebo Kobuki requires at least TurtleBot v2.0

- Problem
  - Feature model captures only variability in space
  - Only exactly one version of variable asset in feature model (no variability in time)
Hyper Feature Models (HFM)s for Versions

**Version-Aware Constraints**

1. Autonomous $\rightarrow$ Detection
2. Keyboard $\lor$ Gamepad $\rightarrow$ Webservice
3. Infrared $[\geq 2.0] \lor$ Ultrasound $\rightarrow$ Detection $[\geq 1.1]$
4. TurtleBot $[\geq 2.0] \rightarrow$ Engine $[\geq$ Kobuki 1.0$]$
5. TurtleBot $[1.0 - 1.1] \rightarrow$ Engine $[\leq$ Create 1.2$]$
6. TurtleBot $[\geq 2.0] \rightarrow$ ? Webservice $[\geq 1.1]$
Configurations of Hyper Feature Models

Version-Aware Constraints
1. Autonomous $\rightarrow$ Detection
2. Keyboard $\lor$ Gamepad $\rightarrow$ Webservice
3. Infrared $[\geq 2.0] \lor$ Ultrasound $\rightarrow$ Detection $[\geq 1.1]
4. TurtleBot $[\geq 2.0] \rightarrow$ Engine $[\geq$ Kobuki 1.0]
5. TurtleBot $[1.0 - 1.1] \rightarrow$ Engine $[\leq$ Create 1.2]
6. TurtleBot $[\geq 2.0] \rightarrow$ ? Webservice $[\geq 1.1]

17.09.2014  Management of Variability in Space and Time in Software Families - Christoph Seidl
Variability Model for Variability in Time

Variability Model

- Feature Models
- Hyper Feature Models
- Version-Aware Constraint Language

Variability Realization Mechanism

- Delta Modeling

Variant Derivation Process

- Transformation
- 2 Extension for Variability in Time
- 3 Extension for Variability in Time
Variability Realization Mechanism for Variability in Time?

Variability Model
- Feature Models
- Hyper Feature Models
- Version-Aware Constraint Language

Variability Realization Mechanism
- Delta Modeling
- Extension for Variability in Time

Variant Derivation Process
- Transformation
- Extension for Variability in Time
Delta Modeling: No Variability in Time (Yet)

- Delta Modeling (currently) handles only variability in space
- Delta Modeling is a form of transformation
  - May be extended for variability in time
    - Explicit notion of evolution delta modules
Evolution Delta Modules are Different

1. Intent
   - Creating Variants vs. Meeting New Requirements

2. Predictability
   - A Priori Known vs. Unplanned

3. Expressiveness
   - Intentionally Limited Expressive Power vs. Great Expressive Power
Example: Using Evolution Delta Modules

```java
public class Movement {
    void handleEvent(Event e) {
        //Lengthy code here...
    }
}
```

1. Rename class
2. Extract method
   a) Create new method
   b) Move code to new method
   c) Call new method

```java
public class MovementController {
    void handleEvent(Event e) {
        handleKeyboardEvent(e);
    }
}
```

Evolution Delta Operations

- Configuration Delta Operations
- Modifying Identifiers
- Refactorings

```java
public class MovementController {
    void handleEvent(Event e) {
        handleKeyboardEvent(e);
    }
}
```

```java
private void handleKeyboardEvent(Event e) {
    //Lengthy code here...
}
```

17.09.2014  Management of Variability in Space and Time in Software Families - Christoph Seidl
Variability Realization Mechanism for Variability in Time

- Feature Models
- Hyper Feature Models
- Version-Aware Constraint Language
- Delta Modeling
- Evolution Delta Modules
- Transformation

3

Extension for Variability in Time
Variant Derivation for Variability in Time?

Variability Model
- Feature Models
- Hyper Feature Models
- Version-Aware Constraint Language

Variability Realization Mechanism
- Delta Modeling
- Evolution Delta Modules

Variant Derivation Process
- Transformation
- Extension for Variability in Time

17.09.2014
Management of Variability in Space and Time in Software Families - Christoph Seidl
Features are mapped to configuration delta modules

Versions are mapped to evolution delta modules
Ordering Delta Modules

- Common case
  - Dependencies of delta modules specified explicitly
  - Topological sorting to determine application order

- With mapping of features/versions to delta modules
  - Relation of versions to features (version containment in HFM)
  - Relation of versions (development lines, branching etc.)

- Strict ordering of delta modules allows assumptions on previously applied delta modules

- Specify default ordering of delta modules associated to HFMs
I. Configuration delta modules
II. Evolution delta modules in order of development lines
III. Continue for rest of HFM
Delta Module Order

1. \(c\delta \text{TurtleBot}\)
2. \(e\delta \text{TurtleBot@1.0}\)
3. \(e\delta \text{TurtleBot@2.0}\)
4. \(e\delta \text{TurtleBot@2.1}\)
5. \(c\delta \text{Engine}\)
6. \(e\delta \text{Engine@1.0}\)
7. \(e\delta \text{Engine@1.1}\)
8. \(e\delta \text{Engine@Kobuki 1.0}\)
9. \(c\delta \text{Movement}\)
10. \(e\delta \text{Movement@1.0}\)
11. \(c\delta \text{Keyboard}\)
12. \(e\delta \text{Keyboard@1.0}\)
Complex Mapping to Delta Modules

- Logical expressions of features and versions mapped to delta modules
- Use expressions from version-aware constraint language
- Subsume simple mappings
- Potential dependencies have to be expressed explicitly
- Topological sorting using this information
Applying Delta Modules

Software Variant A

Contains version information!

Software Variant B
Variant Derivation for Variability in Time

Variability Model

Feature Models
Hyper Feature Models
Version-Aware Constraint Language

Variability Realization Mechanism

Delta Modeling
Evolution Delta Modules

Variant Derivation Process

Transformation
Feature and Version to Delta Module Mapping
Implicit Delta Module Evaluation Order
Conclusion

Variability Model
- Feature Models

Variability Realization Mechanism
- Delta Modeling

Variant Derivation Process
- Transformation

Variability in Space
- Hyper Feature Models

Variability in Time
- Version-Aware Constraint Language

Evolution Delta Modules

Feature and Version to Delta Module Mapping

Implicit Delta Module Evaluation Order
Delta modeling is an approach to handling variability in a family of similar yet different software applications, which can handle variability of functionality, evolution and end user customization uniformly. Currently, delta modeling faces the problem that the majority of delta languages is built upon proprietary data models, which results in extensive work when making a source language delta-aware and having to create required tooling manually.

Delta Ecore provides delta modeling on basis of the structured data models concept. The approach allows reuse of many shared functions across different modules etc. as Ecore comes with a variety of tools.
Thank you for your attention!

Questions, Comments, Feedback?