Variability Implementation for Software Product Lines

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

SPL Variability

- To implement software product lines, the underlying code has to be variable
- Variability is the ability to derive products from a common set of artifacts

Dimensions of Implementation

- There are several techniques to implement variability
- Implementation techniques can be classified in three dimensions
  - Binding time
  - Language vs. Tool-based
  - Annotation vs. Composition

Binding Time Classification

Binding Time Techniques

- Implementation techniques allow binding decisions at different times
  - Compile-time binding (static variability)
  - Load-time binding
  - Run-time binding (dynamic variability)

Compile-time Binding

- Developers make decisions of which features to include at (or before) compilation
  - Code of deselected features is not compiled into the product
- Examples of compile-time binding
  - Preprocessors
  - Feature-oriented programming

Load-time Binding

- Developers decide about feature selection when the program is actually starting
- Load-time binding can use command-line parameters or configuration files
  - Parameter-based programming

Run-time Binding

- Features can be enabled or disabled during program execution
  - Reconfiguration is triggered by external or internal stimuli
- Example of run-time binding
  - Context-oriented programming

Why Compile-time Binding?

- Compile-time binding leaves room for optimizations
  - Code can be actually removed from a product
- Load-time and run-time bindings may incur a memory and performance overhead

Why Run-time Binding?

- Load-time and run-time bindings are more flexible to reconfigure
- However, load-time binding requires restart the product

Language vs. Tool-based

Languages and Tools

- Variability implementation can be supported by programming languages and tools
  - Language-based techniques
  - Tool-based techniques

Language-based Techniques

- These techniques use mechanisms provided by a host programming language
  - Implementation of features and variability management are located in the source code
- Examples
  - Aspect-oriented programming (AspectJ)
  - Feature-oriented programming (AHEAD)

Tool-based Techniques

- These techniques use one or more external tools to implement features
  - They favor a clear separation between feature implementation and variability management
- Examples
  - FeatureIDE
  - Colored IDE (CIDE)

Annotation vs. Composition

Annotation and Composition

- Two different ways to represent variability in the code are
  - Annotation
  - Composition

Annotation-based Techniques

- They annotate a common code base (feature)
  - During product configuration, code belonging to deselected features is removed or ignored
- Examples
  - Preprocessor
  - Parameter-based (e.g., if statement)

Composition-based Techniques

- They implement features as composable units
  - Ideally, one unit per feature
  - Unit can be a file, container, or module
- During product configuration, units of selected features are composed in the final product
- Examples: plugins, FOP, and AOP

Annotation vs. Composition

- Main issues
  - Annotation-based techniques are often criticized for their complexity, lack of modularity, and poor readability
  - A key challenge for composition-based techniques is to keep the mapping between features and units
- A combination of annotation-based and composition-based techniques is possible

Bibliography

- S. Apel, D. Batory, C. Kastner, G. Saake. **Feature-Oriented Software Product Lines: Concepts and Implementation**. Springer; 2013.
  - Section 3.1