The Problem

Since J2SE introduction in Java 5, annotations have become a major feature in enterprise Java development, as a powerful tool for extending Java with user-defined metadata. Annotations allow for adding metadata to classes, fields, methods, and parameters at runtime.

Encapsulation

Composition

Polymorphism

The ability to replace multiple, generic annotations with one single, declarative, repeatable annotation can help to address a key issue in dynamic runtime behavior.

Guidelines

Future Work

Architecture & Control Flow

Getting It Right

If you are writing in an OS that supports annotation processing as part of its build lifecycle, as most of the currently popular IDEs do, consider using annotation processors. However, any existing code that uses regular annotations will continue to work, and you can enable (or disable) annotation processing on a per-class basis.

AspectJ

Java implementation

Bytecode Instrumentation

By introducing the Composite annotation, we can achieve the desired behavior, as encapsulation allows for a more flexible and maintainable approach to managing annotations.

The Composite annotation can be used to define a hierarchy of annotations, where each annotation can have its own set of sub-annotations. This hierarchy can be used to define complex annotation patterns, such as composite annotations.

Using @Composite

Java implementation

AnnotatedElements client interface

Copyright © 2009, Xebia

Encapsulation

Composition

Polymorphism

Guidelines

Future Work

AspectJ

Copyright © 2009, Xebia

Robert van Elsacker

Encapsulation

Composition

Polymorphism

Guidelines

Future Work

AspectJ

Copyright © 2009, Xebia

Robert van Elsacker