# Exploring The Language

*draft June 11 2005*

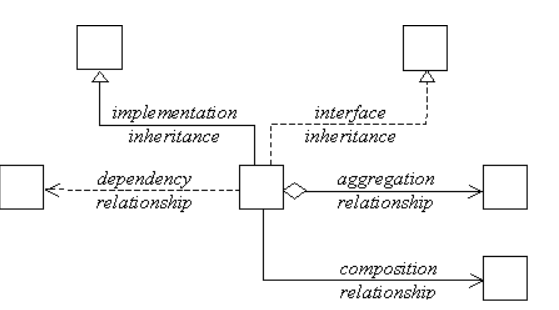
[sven.gorts@refactoring.be](https://web.archive.org/web/20070820101358/mailto:sven.gorts@refactoring.be)

*Abstract: Before we can use thumbnail sketches effectively we need to understand the language's notation and assumptions.*

When drawing thumbnail design sketches during a conversation, all of the language's notation and assumptions can be explained when the need for drawing more complex diagrams arises. At whiteboard discussions and especially when pair-programming the interactive way of learning is quite effective. Unfortunately, an article is a much more limited form of communication. In this article we'll try to make up for that by taking a brief tour exploring the thumbnail language.

## Classes and Their Relationships

As a language intended to express design evolution the thumbnail language inherits most of its notation from UML class diagrams. Refactoring thumbnails are a bit more abstract than class diagrams. Classes are denoted as small rectangles, typically omitting the class name and class compartments. The idea is to look at a design as a sequence of pictures taken from a birds-eye view. The notation for relation between classes is entirely borrowed from UML class diagrams. The figure below gives a quick overview.



*classes and their relationships*

In fact, thumbnail sketches are simplified class diagrams. Any fields and methods are omitted, in favor of a more abstract notion, features. As we will see in the next paragraphs the language distinguishes between three kinds of features: duplication, variation and dependencies. For now, think of a feature as a meaningful software entity in that can range from as small as a constant, including fields and methods, to as large as a class. We'll have a more detailed discussion about features later.

## Duplication

Because thumbnail sketches are miniature class diagrams, any features of importance need to be highlighted using a compact notation. We'll keep the notation dense and simple by denoting features. Whether a feature is implemented by a method or a field, perhaps both. It doesn't really matter, we just denote it as . When we restrict ourselves to the features relevant for a given refactoring a thumbnail class might look something like this:



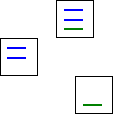
*a class with two features*

Within a class rectangle, any highlighted feature is defined to be unique within its class. Intended for describing large refactorings the language makes no attempt to express code duplication that remains local to the class. However, among different class rectangles the highlighted features do represent duplication as expressed below.



*classes with duplicated features*

Most of the time we won't really care exactly what features are being duplicated, all we want to express is that the duplication exists and need to be dealt with somewhere in the course of the refactoring. In the rare cases where certain clusters of duplication need to be associated we may do so using colors.



*related duplication in different classes*

|  |
| --- |
| In the thumbnail language the use of colors is arbitrary and essentially meaningless. Nevertheless colors are often a great way to extend the expressiveness of a sketch. Use colors freely. |

## Variation

For most large refactorings we will primary focus on similarities, the code duplication let's say. Similarities, although convenient are just one way to look at the code. Another approach that is often useful is to look at the differences. With differences we mean of course features. Features that are different by implementation and thus form a source of variation in the code. We'll denote such a varying feature like .



*a class with two features*

By itself the meaning of a class rectangle with two varying features is not different from the duplication notation. Each of the carved lines represents a unique feature, and that's it. Explicitly denoting variation however, is convenient when we want to highlight the differences within similar code. In the example below the two class rectangles represent classes that differ at three feature implementations but are otherwise identical.



*classes with variated features*

The motivation for the thumbnail language to use different symbols for duplication and variation is primarily a matter of notation and focus. Indeed, duplication and variation are only different ways to look at a given situation. If we have two classes, for example, with code that is for about 80% identical this means that the code is 20% different. The sketch below clarifies this by mixing the two notations.



*80% identical, yet 20% different*

Drawing thumbnail sketches I rarely use the mixed form. Notable exceptions being, situations where I'm explaining the concepts of duplication and variation to colleague developers.

## Dependencies

Other than duplication and variation a third force that often also plays an important role in large refactorings are dependencies. While most dependencies can already be expressed with the relationships inherited from UML class diagrams the language foresees a special purpose notation to highlight dependencies that are of critical importance in a refactoring context. Classes, for example that suffer from compile or runtime problems due to a critical dependency are expressed as follows.



*a critical dependency*

Critical dependencies, are typically dependencies on libraries or resources that may not be available at any time during your development process. Examples include, database connections, socket communications to other processes, shell environment calls or direct calls to a library that has some of these problems. Usually critical dependencies are run-time dependencies, but the notation may be used for compile-time dependencies as well.

## Summary

We've briefly explained the basic elements of the thumbnail language, which inherits most of its notation from UML class diagrams. Thumbnail sketches are a bit more abstract than class diagrams: classes are drawn as small rectangles and notation of their properties is limited to features. The feature notation distinguishes between duplication, variation and dependencies.