Visualizing Class Interfaces with Formal Concept Analysis

Udi Dekel, ISRI - School of Computer Science, Carnegie Mellon University
Yossi Gil, Department of Computer Science, Technion - Israel Institute of Technology

Why do programmers "reinvent the wheel"?

One reason is the difficulty in mastering classes with wide interfaces

Most tools display methods and fields in alphabetical lists rather than in groups of related functionality

We rely on the use of fields by methods to heuristically group related methods and display them in a concept lattice

Lattice operations are used to examine the class at different levels of abstraction

Lattices can be used within a three-stage methodology for investigating class interface, structure and code

In preliminary user studies, developers easily mastered the lattice and discovered errors in real classes

We built prototypes and are now exploring applications for class design, metrics and version comparisons

The safety and portability of modern OO languages like Java should have made reuse of classes commonplace. Yet, programmers are still wary of using third-party classes and often devote significant resources to reimplementing the same abstractions.

One designing factor is the effort required for learning to use a third-party class, whose interface can be impenetrable, often spanning hundreds of methods. It is difficult to intuitively locate specific functionality, especially due to terminology differences. This complexity also increases the risk of interface mismatches and implementation errors.

Sun's Javadoc and most code-browsing and development tools display class members in alphabetical lists, making it difficult to locate methods with related functionalities. For example, the figure to the right lists the methods of a class representing chemical molecules. Can you easily find all the methods for counting the neighbors of each atom, or tell if it is possible to give several names to a certain chemical and then remove some of them?

The premise of our work is that the patterns of field use by methods are functional to understanding the class, and that methods which use the same fields are likely to be semantically related. Hence, we employ mathematical Formal Concept Analysis to heuristically partition the methods of a class into groups of related functionality, and dispel them hierarchically in a concept lattice. In the lattice to the right, each concept (node) represents a set of methods which use the same combination of fields. The intuition of the set of fields of that concept with those of all the concepts below it.

To examine the class at different levels of abstraction, various techniques are used to expose more information. A developer can use them to obtain an instant bird's-eye view of the class before delving into details of specific functionalities. Some informal techniques are presented in the lower figure.

The concept lattice and abstraction techniques form the basis for a detailed three- stage methodology for studying classes: first, the class interface is studied using a concept lattice that includes only public members. Next, we add non-public members, together with documentation about them, to the lattice. Finally, we add fields as follow, if we have access to the source code, the lattice is used to assert an effective inspection order.

Our technique was applied to several real-life case studies. In some classes, studying the lattices led to the discovery of new errors which were difficult to detect with traditional tools like the unaided developer's eye. Preliminary user studies confirm that even novice programmers are able to rapidly understand and use our lattices, and were able to discover many of the same errors that we found.

We built several software prototypes that implement our techniques, including an Eclipse plugin and a batch analyzer for processing entire libraries. We are now exploring the use of concept lattices in CASE tools, where users will interactively add members to a lattice instead of to a linear list in the class diagram. Other avenues of research include traditional version comparisons, and the development of a lattice-based class metrics suite.

Our concept lattice presents a convenient structured view of the class interface

The field-accesses relation is a heuristic for an automatic feature categorization

Lists of Methods and Fields
(Sorted alphabetically as in JavaDoc)

Concept Lattice of the Molecule Class
(Methods are clustered in concepts according to their use of fields. A method uses all fields in its concept and those below it.)

Which representation of the class above is more convenient? How many problems can you spot?

The class can be represented at different levels of abstraction

For more information on this research, please visit http://www.urdidekel.com/research