Interactive Exploration of Structural Concepts in Code

Paul Heckmann and Daniel Speicher
Institute of Computer Science III, University of Bonn, Germany
heckmann@uni-bonn.de, dsp@cs.uni-bonn.de

Abstract. Understanding a software system is the first task in any reengineering activity. For this very challenging task one effective approach is to identify interesting and reoccurring structures in the software and to study these structures individually. In object-oriented software such structures typically consist of a few classes. The well known among them are called design pattern. Yet, which structures to look at in particular? Can we identify interesting structures that are not that well known? Which structures to be a clue to start with?

In this paper we extend a previously suggested approach of pattern mining using Formal Concept Analysis. We propose a way to eliminate redundant information in the overall analysis result. Besides that, we introduce two new features: The first feature is a filtering element that allows us to interactively and dynamically narrow the analysis space. The second is the prominence of a class - a measurement of the importance of the class to the overall system.

In an experimental evaluation we applied our approach on two software projects. In the first, JUnit, our tool guided the experimenter to central structures that can be found in the online documentation yet was unknown to the experimenter. In the second the tool led us to core structures of our own software.

Keywords: Design Pattern, Pattern Mining, Formal Concept Analysis, Interactive Software Exploration.

1 Introduction

An object oriented software system essentially can be seen as a composition of structural concepts. Concepts in which classes and interfaces are connected with each other using building mechanisms like abstraction, inheritance, and composition to realize a certain functionality important to the respective part of the system. Some of these concepts reoccur over the entire project and constitute to a programs unique character, others reoccur yet only in certain parts suggesting core concepts of the program. Hence revealing any of these structural concepts can be a first important step to understanding the software itself, its character and its core functionality. But not all of these concepts arise by design, e.g. by using design patterns as introduced by [1]. They may arise implicitly and strongly depend on the developers style of solving a certain design problem.

In this work we propose an approach to mine structural concepts using a bi-clustering technique called Formal Concept Analysis (FCA) [2], building on a previous approach proposed by [3]. This technique allows us to group structures in source code into meaningful groups without requiring any knowledge on the to-analyze program nor the existence of a reference library of structures. We then improve this approach by the use of
a more efficient mining algorithm and extend it by adding filtering features that, on one hand, allows us to interactively explore the structures in a program, and on the other hand supports us in finding those structural concepts constituting to its core functionality.

In section 2 we give a brief introduction to the very basic idea of FCA. In section 3 we reproduce the approach firstly introduced by [3] to apply FCA on our problem of mining structures in source code. In section 4 we present our extensions to this approach. Finally, in section 5 we validate the performance improvement on three software projects of different size and conduct two experiments to examine the practicability of our extensions.

2 Formal Concept Analysis

Formal Concept Analysis (FCA) [2][4] is a branch of lattice theory that allows us to identify meaningful groupings of objects $G$, i.e. quantities in a data set, that have common attributes $M$. In all the extent of this work, we are going to explain FCA on a very simple yet illustrative example in which we pick a set of birds as $G$ and a set of bird characteristics as $M$. The triple $(G,M,I)$ is called a formal context, where $I : G \times M \rightarrow \{0, 1\}$ is an incidence function which returns 1 for a pair $(g,m) \in G \times M$ if $g$ has attribute $m$, 0 otherwise. This formal context can be organized in an incidence matrix $M$, as depicted in Table 1. Here, $M_{ij}$ has an entry if object $I(i,j) = 1$.

Using this context, FCA groups the objects and their attributes into formal concepts, listed in Table 2. Such a formal concept consists of two sets, an extent and an intent. The intent contains all common attributes that apply to the objects in the extent. In the same way all objects contained in the extent share all properties contained in the intent. Therefore a concept is a maximal collection of elements sharing common properties. Adding an attribute to a concept’s intent there would be at least one object in the extent that does not have this attribute. Adding an object to the extent there would be at least one attribute in the intent this object does not have. As a consequence the formal concepts build a complete partial order that can be written as a lattice. Table 2 in some way suggests this order by the increasing number objects and the decreasing number attributes from top to bottom.

1 It needs to be noted that ornithology usually is not part of our research. The data shown in Table 1 may not be entirely correct.

2 In the context of this particular work, we are not making any use of this partial order yet.