ConAn: A tool for the Identification of Crosscutting Concerns in Object Oriented Systems based on Type Hierarchy Analysis

Mario Luca Bernardi, Giuseppe Antonio Di Lucca
Department of Engineering - RCOST
University of Sannio
Benevento, Italy
{mlbernar, dilucca}@unisannio.it

Abstract—In Object Oriented systems the analysis of Type Hierarchies allows to identify class members that are scattered and tangled along each hierarchy, thus contributing to the implementation of static crosscutting concerns. This paper presents ConAn: a tool to automatically analyse the Type Hierarchies in an existing system and to identify the Type Fragments (i.e. a portion of a Type in terms of its members and relationships) implementing static crosscutting concerns. The structural information about the Type Fragments composing each concern and the crosscutting relationships among them are useful to drive their re-engineering towards Aspects.

Keywords: Reverse Engineering, Aspect Mining, Aspect Oriented Programming, Software Evolution, MOF

I. INTRODUCTION

In OO systems, the usage of Types Hierarchies (and mainly the ones rooted in Abstract Classes or Interfaces) is a way introducing scattering and tangling in the modules implementing any Type along the Hierarchies. As an example, a method declared by an Interface and implemented by more than one class will have its behaviour scattered in all the classes implementing it and tangled with the behaviour of the other methods implemented in each class. A main reason for that is the fact that usually inheritance, interfaces’ implementations and Type nesting, are a way to implement crosscutting behaviours (i.e concerns) by super-imposition: each Type in the system that need to contribute to a certain concern is forced to implement an interface, to inherit or to contain another Type.

Each Type introducing a new external behaviour (e.g. a new method) can be (initially) associated to a separate concern; we call such a Type the 'Seed' of the concern. Concerns can be represented as sets of Type Fragments, by the MOF (Meta Object Facility) meta-model proposed in [1], where a Type Fragment is a portion of a Type made by (some of) its attributes, methods, and its relationships (inheritance, implementation, and containment).

The static analysis of the system’s Type Hierarchies allows to identify the concerns’ Seeds and the related Type Fragments participating in concern implementations. Since the focus is on the static crosscutting, the analysis is performed at member/method grain, i.e. methods are seen as black-boxes and their internal code is not analyzed but just their declarations (the signatures) are considered. This means that the lowest Fragments’ grain we consider is at level of a whole method, not considering its internal blocks of code. Seeds can have a granularity lower than concerns, i.e. a concern can be referred to more Seeds each one implementing that concern in a different context. All the Seeds that contribute to implement the same concern can be grouped together and each of Seed-group, along with the Type Fragments associated to them, is assigned to a single more abstract concern. In this way, the concerns of the system are identified together with all the Type Fragments making up them. The crosscutting among the concern is identified by looking for scattering and tangling of the Type Fragments within the identified concerns.

II. THE CONCERN ANALYSER TOOL: CONAN

The “Concern Analyser” (ConAn) tool supports and automates a process to analyse the Type Hierarchies and identify the crosscutting concerns in Java systems [1]. The process is based on four main steps: (1) Analysis of Type Hierarchies to find concerns’ Seeds; (2) Type Fragments Identification and MOF model instance generation; (3) Clustering of Seeds; (4) Crosscutting Analysis.

The ConAn tool was developed on top of the Eclipse platform and exploits the JDT tools. The Figure 1 shows the tool architecture. The tool is composed of three main components: Model Handler (MH), JDT Extension (JDTE) and, on top of these ones, Presentation Layer (PL). The JDTE adds to the JDT Core components the functionality needed to build a complete structural representation of a Java system taking into account the Types and their members, at the grain of fields and whole methods, and their relationships (inheritance, implementation and Type nesting). The MH uses the JDTE and EMF components to generate an instance of the MOF meta-model. The MH performs a traversal of the Type hierarchies of the system to extract the information needed to find the seeds and to instantiate the meta-model and associate a concern to each identified seed. Any Reference Type (i.e. any seed) that introduces, in all paths from a root to a leaf of the Type hierarchy graph, the declaration of a new set of members is detected. Each seed is associated to a concern just on the base of the extracted structural information regardless of any semantic meaning. At the end of the traversal, each concern is associated to the Fragments of all Types that implement the corresponding seed.

ConAn provides both semi-automatic and automatic support for the clustering of seeds. The automatic one exploits...
the Hierarchical Agglomerative Clustering (HAC) described in [1]. In the semi-automatic case, the user will select which groups or seeds realize a single (more general) concern and give it a name and a description. The user can modify/refine the groups defined by the automatic HAC by "merge" and "split" operations, on instances of the meta-model, provided by the Model Handler and Model Editor.

The figure 2 shows a snapshot of the ConAn’s GUI, where several views, implemented in the Presentation Layer and grouped into an Eclipse Perspective, are reported. On the bottom of the Figure 2, the Concern Model Editor reports the tree structure of the meta-model instance. It is a customization of the default editor by the Eclipse EMF Tools from the meta-model MOF specification, whose some operations were added to: support the seed merge/split operations; compute metrics such as the number of fragments and the members belonging to each concern, and the Degree of Scattering for Classes (DOSC) [2]; and to generate a UML2 class diagram modelling the internal structure of a concern (the window on the left-bottom side). This allows a user to gain knowledge about the concerns’ structure and to make comparisons among the concerns (e.g. to find the most crosscutting concerns’ members in each Type).

On the bottom-right side of Figure 2, the Crosscutting Matrix is showed. This is a symmetric matrix on whose rows and columns the concerns are reported. It reports, for each pair of scattered concerns, the number of Type Fragments generating tangling, i.e. a matrix value $M_{ij}$ represents the number of fragments that are tangled between the Concern in row $i$ and the one in column $j$. By clicking on a cell, a list of the involved fragments is shown and selecting one of them it is located in the Model Editor.

In the upper side of the Figure 2 the windows representing the Crosscutting View are shown. The Crosscutting View, is a customized version of the Eclipse Visualiser. It provides information about the crosscutting at the member grain. This view, represents Types (e.g. classes, interfaces), or packages, as boxes and each line in a box represents a member. Each colour represents a concern and each line has the colour of the concerns the member is a part of. The boxes and the lines are linked to the sw components: a double-click on them allows to open a window where the Types and members associated to a concern are edited (both Java and Model editors are used). A user can select the concerns of interest by using the Visualizer Menu shown on the upper-right side in the figure.

III. CONCLUSIONS

Several open source java software systems were analysed by ConAn. The main analysed systems were (by LOC size) XWorks, JHotDraw 6.x and 7.x, OSWorkFlow, FreeMind, and JFC. In all the systems, Types along hierarchies, and in particular all the interfaces (as expected), were found as seeds of concerns. The results from ConAn were validated by comparing them with the ones resulting by manual code inspection carried out by one of the authors, considered as a 'gold standard'. The percentage of 'bad concerns', i.e. concerns whose composition by Type Fragments was different from the 'gold standard', was not greater than 7%. The manual inspection required, in all the cases, a greater effort with respect the usage of ConAN: about an average of the 43% of more time was required. This showed the utility, effectiveness and efficiency of ConAn. Future work will be mainly addressed to consider fragments of code inside methods in order to identify also dynamic crosscutting.

REFERENCES
