The ConAn Tool to Identify Crosscutting Concerns in Object Oriented Systems

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

Abstract—This paper presents the main features of ConAn, a tool supporting an approach to find scattered and tangled class members in OO systems and to group them in concerns. The recovered information is useful for refactoring/migration tasks, such as towards Aspect Oriented Programming (AOP).

I. INTRODUCTION

The evolution/migration of existing OO systems towards Aspect Oriented Programming (AOP), allowing a better separation and encapsulation of the (crosscutting) concerns, is a way to eliminate, or reduce at a minimum, the scattered and tangled code implementing the crosscutting concerns. A first step to efficiently and effectively drive the re-engineering towards AOP is the automatic identification of the code components implementing the crosscutting concerns in a system. ConAn provides an automatic support to identify the code components (at class and method level) scattered and tangled along the type hierarchy of an OO system. The present version of ConAn extends the one in [1]. The main improvement is the analysis of the methods’ call graph to identify more and new crosscutting relationships among the identified concerns. ConAn performs a static analysis of the source code of a Java system according to the approach defined in [2]. The approach defines a meta-model to represent concerns at class level as sets of Type Fragments (i.e. a portion of a Type in terms of its members and relationships). Each Type introducing a new external behavior is associated (initially) to a separate concern; such a Type is said the “Seed” of the concern. To identify both the Seeds and the Type Fragments participating in Seeds implementation an analysis of the type hierarchy of the system is performed. A concern (e.g. Persistence) can be referred to more Seeds, each Seed implementing that concern in a different context (e.g. the Persistence of different specific objects). All the Seeds that contribute to implement the same (general) concern can be clustered together and each cluster of Seeds, along with the Type Fragments associated to them, is assigned to a single more abstract concern (e.g., all the Seeds referred to the Persistence of different specific objects, can be clustered in one Persistence concern). The crosscutting among the identified concerns is found by looking for: (i) the Type Fragments scattered and tangled within the identified concerns, due to the static structure of declarations specifying the system’s Type hierarchy; and (ii) the calls among methods belonging to Type Fragments assigned to different concerns. The latter case is the main extension of the current version of ConAn with respect [1]. In this case, given two concerns $C_x$ and $C_y$, a crosscutting relationship is between them if and only if: (i) in the system there is a method containing at least a call towards $C_x$ and another towards $C_y$ and (ii) there are (scattered) calls in different methods towards at least $C_x$ or $C_y$. ConAn performs an analysis of the methods’ call graph to identify the Inter-Fragments calls involving different concerns and satisfying the above condition. Crosscutting relationships due to the first case are identified and represented by a Crosscutting Matrix. This is a symmetric matrix whose rows and columns correspond to the identified concerns. 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 $j$.

II. THE CONAN TOOL

The ConAn tool was developed on top of the Eclipse platform and exploits the JDT tools. The Figure 1 (a) shows the tool architecture. The JDT Extension (JDTE) component adds to the JDT Core components the functionalities needed to build a complete structural representation of a Java system taking into account the Types, their members (at the grain of fields and whole methods) and their relationships (inheritance, implementation and Type nesting). The Model Handler (MH) uses the JDTE and EMF components to generate an instance of the defined meta-model. The MH performs a traversal of the system Type hierarchy to extract the information needed to find the seeds, to instantiate the meta-model, and to associate a concern to each identified seed [2]. 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. The InterFragment Calls Handler (IFCH) builds, using a Rapid Type Analysis (RTA), a context-insensitive call graph of the entire system. The calls relationships are traced to the Type Fragments (i.e. the identified Concerns) and the MOF concern model instance is updated according to such information. The Clustering component supports the clustering on Seeds; both manual and semi-automatic support [2] is provided at this aim. The user will give the clustered concerns a (meaningful) name and a description (this may require a manual inspection of the internal fragments of each concern resulted after clustering). Manual refinement on concerns associated to Seeds
The architecture of ConAn (left) - A snapshot of its User Interface (right).

is performed by means of “merge” and “split” operations provided by the ConAn’s Model Editor.

The figure 1 (b) shows a snapshot of the ConAn’s GUI, where several views, implemented in the Presentation Layer component and grouped into an Eclipse Perspective, are reported. In this Figure the Concern Model Editor reports the tree structure of the concern model allowing to:

- browse the system’s concerns inspecting their internal structure (in terms of Fragments);
- execute merge/split operations on concerns and their members;
- compute metrics, such as the number of fragments and the members belonging to each concern, or the Degree of Scattering for Classes (DOSC);
- generate a UML2 class diagram modelling the internal structure of a concern.

This allows ConAn’s users to gain more easily the needed knowledge about the concerns’ structure and helps them to make comparisons among the concerns (e.g. in terms of degree of crosscutting).

The window with the Crosscutting View is a content provider of the Eclipse Visualiser where the content is extracted from the concern model of the selected project. The view shows graphical information about the crosscutting at the member grain. It represents types (e.g. classes, interfaces), or packages, as boxes and each line in a box represents a member. Colours are associated to concerns and each line is painted by the colour of the concerns the member is a part of. The boxes and the lines are linked to the code components: a double-click on them opens the JDT editor view, allowing a quick access to the concern’s members. A user can select the concerns of interest in the Concern Model Editor or by using the Visualizer Menu.

Another view shows the Crosscutting Matrix. Due to space constraints, this view is not in the figure; anyway a click on a cell of the Matrix allows to visualize the list of the involved fragments and the user can select a fragment in the list to locate it in the Concern Model Editor. A view is to display the InterFragments call-graph, too. In this graph, Nodes correspond to Type Fragments or Concerns, and edges to calls among them. By selecting a node the corresponding code is displayed, while selecting an edge the corresponding call statement in the code is highlighted.

III. CONCLUSIONS

ConAn was used to analyse some Java systems, including JHotDraw, JFC, XWorks.

The results from ConAn were compared with the ones obtained by manual analysis, carried out by the authors and considered as a ‘gold standard’. Some Concerns (about 7%) identified by ConAn had a different TypeFragment composition with respect to the gold standard. Anyway, the manual analysis required a greater effort (about 43% of more time) than the usage of ConAn. The tool extension for the analysis of the InterFragment call relationships allowed the identification of new crosscutting relationships that was not possible to identify just by the structural analysis, making more precise the concern model and improving the comprehension of the system. Future work will be mainly addressed to support automatic source code refactoring towards aspects.

REFERENCES
