Alborz: A Query-based Tool for Software Architecture Recovery *

Kamran Sartipi

Department of Computer Science, University of Waterloo,
Waterloo, ON. N2L 3G1, Canada
ksartipi@math.uwaterloo.ca

Abstract: Alborz is a user assisted reverse engineering tool designed for analyzing and recovering the architecture of a software system in the form of cohesive modules and subsystems. The tool's operation is based on techniques from the area of data mining, pattern matching, and clustering (Figure 1).

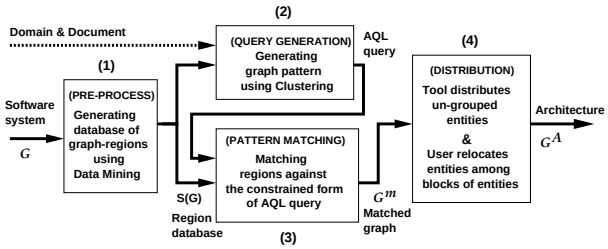

Figure 1. The framework for query-based software architecture recovery.

1 System analysis

The Alborz tool provides facilities for analyzing the software system in order to obtain insight into the system before starting the recovery process. These analyses are based on the notion of inter-/intra-component association (as approximation for coupling and cohesion properties between system files). The *association* is a property among highly related groups of entities in term of the maximum number of shared entities in the groups. This technique allows to measure the modularity of the software system as an indication of the quality of the system design. Also, a *view-based architectural design evaluation model* allows to categorize the system design into different design properties.

The tool provides two complementary techniques for software architecture recovery as followings:

2 Pattern matching technique

In a nutshell, the user defines a graph-based architectural pattern of the system modules (subsystems) and their interactions based on: domain knowledge, system documents, or tool-provided clustering techniques. In an iterative recovery process, the user constraints the architectural pattern and the tool provides a decomposition of the system entities into modules or subsystems that satisfy the constraints.

The user develops a hypothesis about the architecture of the system that can be viewed as a graph of modules and interconnections[1] (i.e., architectural pattern), where each module (one node of graph) represents a group of placeholders for the system entities (i.e., functions, types, variables) to be instantiated, and each bundle of interconnections (one edge of graph) between two modules represents data-/control-dependencies between two groups of placeholders in two modules. The minimum/maximum sizes and the types of both placeholders and the interconnections are considered as free parameters to be decided by the user (respecting the allowed relation between two entities).

This yet un-instantiated module-interconnection representation (can be referred to as *conceptual architecture*) is directly defined for the tool, using a proprietary language that we call *architecture query language* (AQL). Therefore, a query in AQL represents a macroscopic graph-form pattern for a part or the whole of the system architecture to be recovered. The task of the tool is then to search through the software system (again represented as a graph of system entities and relationships) to find an optimal match between the module-interconnection pattern in the AQL query and the graph of the system.

Since finding the exact matching may be impossible, we allow inexact matching by maximizing an association-based

*This work was funded by IBM Canada Ltd. Laboratory - Center for Advanced Studies (Toronto) and the National Research Council of Canada.

[1] Similarly, at the higher level architecture recovery a graph of subsystems (consisting of files or modules) and their interconnections are used.

score function in a *branch and bound* search algorithm. Therefore, we define the architecture recovery as a graph matching problem, in which the matching process searches to find a series of graph edit operations (i.e., node or edge insertion and deletion) with minimum cost that if applied on the expanded form of the AQL graph, the resultant graph matches with a subgraph of the system graph.

Considering the number of entities in a medium size software system (usually more than 1000 entities), searching the whole search space is an intractable problem. Hence, we must restrict the search domain for each module in the query to a group of eligible entities. In doing so, we preprocess the graph of the software system and decompose it into *regions* based on the association property, and then restrict the search for each module to one (or more) of these regions. Therefore, the quality of the recovered modules depends on the proper selection of the region(s) for each module in the AQL query. These regions can be determined by the following clustering techniques.

3 Clustering technique

The tool provides two clustering techniques whose results can also provide an initial graph pattern for the pattern matching technique.

Automatic clustering is a supervised optimization clustering which incrementally recovers the cohesive modules or subsystems from a database of graph regions. This clustering technique also uses AQL query to define a number of uninstantiated modules or subsystems. At any stage of the clustering, a heuristic provides a ranked list of regions for the next module recovery to be chosen by the user, based on: quality of the remaining regions and minimum overlap with already recovered modules. Different similarity metrics such as association among highly related groups of entities, Jaccard metric based on shared functions, types, variable, or all, are supported.

Manual clustering is a visual aid for the user to collect the files or modules into subsystems based on the categorization and mapping of the association values between files (or modules) into colors, which represent the strength of the association between them. A stepwise clustering operation starting from strong association links and continuing with medium, loose, and weak links, allows a manual clustering of the system files into subsystems as well as viewing the structure of the files. A graph visualizer tool such as Rigi is used to view this graph.

4 Tool

The Alborz tool has been implemented in the Refine reengineering environment and uses the Refine's parser generator tool to design the AQL language.

Input/output: the input to the Alborz tool is an information base that corresponds to the entities and relationships of the software system in the form of an AST or RSF file. The tool provides the result of the architectural recovery into two forms: i) HTML pages for the recovered components, tool generated metrics, and source code, to be visualized by a Web browser such as Netscape; and ii) graphs of boxes and arrows to be visualized by the Rigi tool, where the boxes are the analyzed components and the arrows are either the resource interaction (i.e., import/export) between the components or their association strengths.

AQL query: the AQL query defines the pattern that is to be matched against the information base that represents the software system. Each module of the query uses one or more entities as fixed entities to appear in the result of the recovery, namely *main-seed*(s) which determine the corresponding region(s) to be searched for the module, and *seeds* which just appear in the result without search. In the following a part of an AQL query, consisting of a subsystem S1 of files and its interconnection links to other subsystems is shown:

```
BEGIN-AQL
SUBSYSTEM: S1
    MAIN-SEEDS:        files  e_edit,  e_update
    IMPORTS:
      RESOURCES:       rsrc  ?IR,
                       rsrc  ?R1(6 .. 10) S2,
                       rsrc  ?R2(12 .. 20) S4
    EXPORTS:
      RESOURCES:       rsrc  ?ER,
                       rsrc  ?R3(10 .. 15) S2,
                       rsrc  ?R4(1 .. 5) S3
    CONTAINS:
      FILES:           file  $CFI(7 .. 10),
                       files  e_edit,  e_update
    RELOCATES:    NO:
                       files  e_allign,  u_scale    TO:  S3
END-ENTITY
```

The above AQL fragment is interpreted as: the subsystem S1 which will be instantiated with seven to ten files, and definitely contains the files *e_edit* and *e_update* (main seeds), imports minimum six and maximum ten resources (?R1) from subsystem S2. A similar interpretation holds for the *EXPORTS* and *CONTAINS* sections. The notations *?IR* and *?ER* in the import and export parts denote unidentified quantities of links between the current subsystem and any other subsystems in the query that have not been specified by the architectural pattern, therefore, are not matched by the matching algorithm.

So far, we have experimented with systems such as: Xfig, CLIPS, Bash, Apache, and Weltab.

Acknowledgment: I would like to thank Dr. Kostas Kontogiannis for his supports to build this tool.