



Figure 1: The interactive environment for pattern-based software architecture recovery.

1 Alborz: software reverse-engineering toolkit

I have designed and implemented a reverse engineering toolkit (Alborz) [4], to provide an interactive environment for recovery and evaluating the architecture of a software system as cohesive components, based on the techniques such as: i) pattern-matching (presented in my thesis), ii) partitioning clustering presented in [7], iii) incremental optimization clustering presented in [6], and iv) software design views and evaluation presented in [5]. The current version of Alborz consists of 30 KLOC written in the Refine's language. Alborz has been built using the Software Refinery re-engineering environment (Refine) [3], and uses the Refine's built-in parsers to parse the software systems, and built-in parser generator to design a proprietary language called Architectural Query Language (AQL).

The pattern-matching environment provided by the Alborz tool is illustrated in Figure 1 and consists of two phases off-line and on-line that are discussed below. Chapter 8 of my thesis, presents a comprehensive set of experimentations related to the time/space complexity, accuracy, stability, and quality of the proposed architecture recovery technique. The experimentations are performed on six middle-size industrial systems, namely: i) *Xfig.3.2.3* drawing editor, ii) *Clips.4.20* expert system builder, iii) *Apache.1.2.4* http server; iv) *Bash.2.03* Unix shell; v) *Elm.2.5.6* Unix mail system; and vi) *Ghostview.3.5.8* postscript file viewer.

1.1 Off-line pre-process

In the pre-process phase, illustrated on the left part of Figure 1, the source-code information are extracted from the software system that is written in a procedural language. These information are processed and stored in a database in order to be used for the on-line analysis phase. The stored information allow a programming language independent architectural analysis of a software system. Currently, the software system is either parsed into an abstract syntax tree AST (using the Refine parser for C language) or parsed into an entity-relationship format (namely RSF) using Rigi parser [1]. Based on the abstract domain model defined in Chapter 3 of my thesis the parsed software system is represented as an attributed relational graph suitable for architectural analysis. Using data mining techniques the graph of the system is divided into graph regions which are then stored in a database. The off-line analysis can be very time consuming depending on the number of the system entities and the level of inter-relationship among them, however the off-line operations are performed only once for each system.

1.2 On-line analysis

In the right part of Figure 1, using tool-provided system analysis [6, 7], domain knowledge, and/or system documents, the user develops a hypothesis about the architecture of the system (i.e., conceptual architecture) that can be represented as a “module interconnection” pattern¹ using a query written in the Architecture Query Language (AQL) discussed in Chapter 4 of my thesis. The minimum/maximum sizes and the types of both the modules and interconnections are considered as free parameters to be decided by the user. A query in AQL represents a macroscopic graph-form pattern for a part or the whole system architecture to be recovered. The AQL query is parsed to generate a pattern-graph which is the expanded form of the AQL query. A pattern-graph consists of smaller patterns for the modules that are connected through groups of edges, where hard and soft constrains control the number of node/edge insertions/deletions in the matching process.

The pattern-graph and the selected graph regions from the database are supplied to the graph-matching engine that computes a sub-optimal match between two graphs. The graph-matching engine performs approximate matching by minimizing an association-based cost function in an optimization search algorithm. The result of matching is presented to the user through easily understood information in HTML pages or graphs via graph visualization tools (see the HTML pages through link to the case studies). The user investigates the results, and if needed, he/she expands the pattern by adding more abstract modules to the AQL query and/or constrains the interactions between the modules and iterates the matching process.

Query generation: in the proposed pattern-based architectural recovery the pattern is incrementally generated using: association relation among the system entities [5]; clustering techniques [6]; related domain’s reference architecture; available system architecture

¹Similarly, at the higher level of architecture recovery a pattern of subsystems (consisting of files) and interconnections is used.

document; or consultation with the system developers. The objective in any of these methods is to extract small groups of system entities which represent the core functionality of the modules (or subsystems) in the system architecture. These groups are used to incrementally generate a constrained graph of modules and interconnection links represented by an AQL query. The steps for pattern generation are discussed in Section 8 of my thesis.

Evaluation: the proposed environment provides: i) modularity quality metrics based on inter-/intra-component association [5] and inter-/intra-component connectivity [2] to evaluate the software system and its architecture; and three design views of the system, namely *control passing*, *data exchange*, and *data sharing* views [5].

1.3 Architectural design of Alborz toolkit

The architectural design of the Alborz tool consists of a number of components with simple and well-defined interfaces and with a *pipe and filter* style. Each component (filter) processes its input data in the form of a repository file or a memory data-structure (pipe) and stores the results in another file or data structure for the next component. Figure 2 illustrates the architecture of the Alborz tool (inside a dashed box) and its interaction with the surrounding environment. The contents of the repositories and memory units as well as the interface commands are shown in Figure 2.

Tool interface: the interfaces to the Alborz tool is shown outside the dashed box in Figure 2, and consists of: i) Repository R1 to store the parsed software system as AST generated by the Refine's parser, or RSF tuples generated by the Rigi parser; ii) *Intervista*, a GUI tool in the Software Refinery environment, for formulating the AQL query and launching the commands; iii) Gnu Emacs editor for developing and debugging the Alborz tool; iv) Netscape browser for viewing and navigating the source-code, the results of analysis, and the tool generated metrics; and v) Rigi tool for viewing the graph representation of the system 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.

1.3.1 Components

- *Pre-process component:* during the off-line analysis this component is responsible for: i) extracting the system's entity-relationship data from the AST or RSF format of the parsed software system in the repository R1; ii) generating the system data such as frequent-itemsets, entity domains, and similarity matrix for the system entities; and iii) storing the entity-relationships and system data in the repository R2. The products of this phase have been discussed in Chapter 3 of my thesis. During the on-line analysis this component restores all the stored data in repository R2 into the memory M1 to be used by the on-line analysis.
- *Analysis components:* the major tasks of the tool in the on-line analysis are performed by four analysis components in Figure 2. The operations of the pattern matching component are the subject of my thesis, however the operations of the other components,

i.e., incremental clustering [6], partitioning clustering [7], and software design views and evaluation [5], are not presented in this thesis.

- *Query analyzer component*: this component performs two tasks: i) generates a template AQL query including the tool-suggested main-seeds, default sizes for abstract-components, and unconstrained sizes for abstract-connectors in order to allow the user to tailor the sizes and types and proceed with the on-line analysis; and ii) parses the query file using the AQL grammar and domain-model; checks the semantics of the query information; and dispatches the activation commands to an appropriate component according to the directives in the header of AQL query.
- *Output component*: generates both the HTML pages and the graph representation for the recovered architecture, design views, and the computed metrics.
- *HTML generator component*: produces HTML source-code by annotating the system's source-code files.

References

- [1] Rigi, URL = <http://www.rigi.csc.uvic.ca/rigi/rigiindex.html>.
- [2] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *Proceedings of the IWPC*, pages 45–53, 1998.
- [3] Reasoning Systems Inc., Palo Alto, CA. *Refine User's Guide*, version 3.0 edition, May 1990.
- [4] Kamran Sartipi. Alborz: A query-based tool for software architecture recovery. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC'01)*, pages 115–116, Toronto, Canada, May 2001.
- [5] Kamran Sartipi. A software evaluation model using component association views. In *Proceedings of the IEEE International Workshop on Program Comprehension (IWPC'01)*, pages 259–268, Toronto, Canada, May 2001.
- [6] Kamran Sartipi and Kostas Kontogiannis. Component clustering based on maximal association. In *Proceedings of the IEEE Working Conference on Reverse Engineering (WCRE'01)*, pages 103–114, Stuttgart, Germany, October 2001.
- [7] Kamran Sartipi and Kostas Kontogiannis. A user-assisted approach to component clustering. *Accepted for the Journal of Software Maintenance: Research and Practice (JSM)*, 2002.