Alborz: An Interactive Toolkit to Extract Static and Dynamic Views of a Software System [*]

Kamran Sartipi and Lingdong Ye and Hossein Safyallah
Dept. Computing and Software, McMaster University
Hamilton, ON, L8S 4K1, Canada
{*sartipi, lye, safyalh*}*@mcmaster.ca*

Abstract: Alborz is a multi-view, interactive, and wizard-based software architecture reconstruction and evaluation toolkit that takes advantage of the Eclipse plug-in technology to provide feature extensibility, and uses GXL format to interoperate with other reverse engineering tools. The current version of Alborz toolkit supports static and dynamic views of a software system. For the static view, the toolkit extracts the structure of a software system using wizard-guided forms that allow to define the high-level view of the system. The static view represents abstract components and connectors which are then mapped onto the low-level source graph to find approximate matching within the software system. For the dynamic view, the toolkit extracts high-frequent execution patterns by running feature specific task scenarios on the software system. Subsequently, the implementations of the software features in the source code are identified as a means to evaluate the structure of software. The toolkit will be available as an Eclipse plug-in to serve the software reverse engineering community.

1. Introduction

Alborz was originally designed as a research toolkit for software architecture recovery and evaluation in the Refine re-engineering environment [6] at the University of Waterloo [7]. The Refine environment, as the execution platform of Alborz, had drawbacks in terms of low performance and lack of widely adoption by the reverse engineering community. The new version of Alborz toolkit has been developed at McMaster University as an Eclipse plug-in application [1] that is entirely re-engineered such that only the functionality and features of the old version have been re-used, not its design. This practice generated valuable re-engineering experience within our research team. The current version of Alborz consists of more than 14 packages and 160 Java classes/interfaces. Users can easily extend the functionality of Alborz using standard Eclipse plug-in mechanism. The toolkit interoperates with SHriMP toolkit [10] to visualize the results of the static analysis of a software system. The Alborz project aims to provide useful architectural information through both static and dynamic analyzes. For the static view, the tool extracts a number of components and their interactions, where a component is either a number of system files or a number of system functions and the interactions are defined in terms of imports and exports of software entities at function level. For the dynamic view, the tool maps specific software features (defined through a number of task scenario sets) onto the source code that implement those features, and consequently it allows to evaluate the structure of a software system.

[*]This work was funded by National Research Council of Canada.

2. Related Work

The following static and dynamic analysis tools from the literature are related to the Alborz tool.

Stoermer et al. [9] and Hamou-Lhadj and Lethbridge [3] have surveyed a number of static and dynamic analysis tools, respectively. The closest static analysis approach to our approach is presented by Bunch tool [5] that provides a partitioning method to group software modules (e.g., system files) into a number of clusters. Bunch uses a hill-climbing search to consider different alternatives based on neighboring partitions, where the initial partition is randomly selected. Also, the closest dynamic analysis tools to the Alborz tool are as follows. DRT [2] is a tool for design recovery of interactive graphical applications that provides a means to track the implementation of the user's actions (through scenario execution) in the source code. ISVis (Interaction Scenario Visualizer) [4] is a tool that provides traditional static analysis techniques to extract the software components. Furthermore the tool uses task scenarios to extract and visualize the dynamic interactions among the existing components.

Alborz provides a pattern-based architectural recovery technique to recover highly cohesive components that conform with the input pattern. In the dynamic analysis, the frequent patterns in execution traces are used to map the individual software features onto the corresponding software component, and to evaluate the component's cohesiveness.

3. Underlying Theory

In the following parts, we discuss the employed techniques and underlying theories that support the proposed static and dynamic views of a software system.

Static analysis technique
In the static view, based on domain knowledge, system documents, or tool-provided clustering techniques, the user develops a hypothesis about the architecture of the system that can be viewed as a graph of components and their interconnections. In this context, a component is a subsystem of files, or a module of functions, where each component represents a group of placeholders to be instantiated by the system entities. Also, each bundle of interconnections between two components represents data-/control-dependencies between two groups of placeholders in two components. In this form, a system entity can be a function, an aggregate type, or a global variable to fill a placeholder in the component. The minimum/maximum sizes and the types of both placeholders and interconnections are decided by the user.

This yet un-instantiated component-interconnection representation (can be referred to as *conceptual architecture*) is directly defined for the tool, using wizard-based forms. In this context, a *query* 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 that is also represented as a graph of system entities and relationships, to find a sub-optimal match between the component-interconnection pattern in the query forms and the graph of the system. We use a sub-optimal heuristic of A* optimal search algorithm for the pattern matching engine [8].

In a medium or large size software system it is imperative that we restrict the search domain in a large graph generated by the entities and relationships. In this respect, in Alborz toolkit the graph of the software system is preprocessed and decomposed into a large number of *domains* based on the association property that we obtain from data mining operations, and then we restrict the search for each component to one (or more) of these domains. Therefore, the quality of the recovered modules depends on the proper selection of the domain(s) for each component in the query forms. Each domain is identified by its core entity, namely *main-seed* and every other entity in the domain has a high association with the domain's main-seed. The result of the structural recovery is represented as concrete subsystems / modules and their exact interconnections through hypertext links and navigable views provided by the Eclipse platform.

In the design of the Alborz toolkit data mining is used to extract maximal association relation among a group of highly related system entities such as functions, aggregate data-types, and global variables. Maximal association is a property among a group of entities and their shared attributes such that the group of entities share the maximum number of shared attributes.

Dynamic analysis technique
Alborz provides dynamic analysis of a software system based on frequent patterns of execution traces in a software system, where the patterns represent the common functionality of a number of feature-specific task scenarios. Dynamic analysis takes advantage of the data mining technique *sequential pattern discovery* among the software execution traces. As a critical aspect of this dynamic analysis, the specific features of a software system must be identified by investigating evidences such as: system user manual, online help, and similar systems in the corresponding application domain. A set of relevant task scenarios is selected that examine a single software feature. We denote this set as *feature-specific scenario set*. For example, in the case of a drawing tool all scenarios that examine the *move figure* operation of the tool would constitute such a feature-specific scenario set. The software under study is instrumented by inserting particular pieces of code into the software system (source code/binary image) to generate a trace of the software execution. Therefore, running the feature-specific scenarios against the instrumented software system generates a set of execution traces with very large sizes. To make the large size of the generated traces manageable, in the preprocessing step we remove all redundant function calls caused by the cycles of the program loops. The trimmed execution traces are then fed into the execution pattern mining engine to reveal the common sequences of function invocations that exist within the different executions of a program. Each execution pattern is a candidate group of functions that implement a common feature within a scenario set.

Finally, we assign the extracted execution patterns to the system's structure, i.e., files of the system and define two software quality measures to assess: i) local structural cohesion for each software module; and ii) functional scattering of a specific software feature among software modules.

4. Tool Architecture

Alborz toolkit has a data-centric and user-interface driven architecture consisting of six collaborative components, as: User Interface (UI); Datastore; Input/Out (I/O); System Analysis; Pre-processing; and Evaluation. Figure 1 illustrates the hierarchy of Alborz components and subcomponents within the context of Eclipse plug-in mechanism. A key characteristic of the Alborz toolkit is its ex-

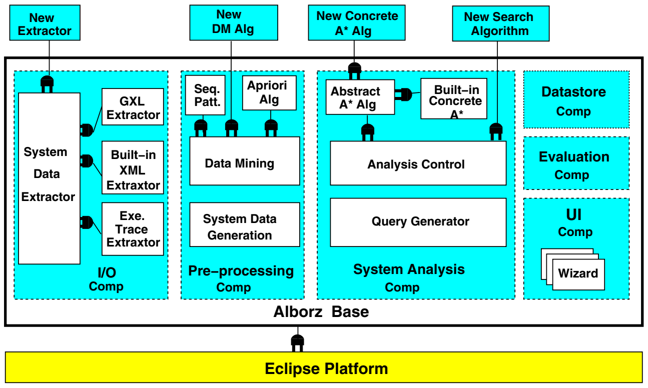


Figure 1. Structure of Alborz components in the context of Eclipse plug-in mechanism.

tensibility by leveraging the power of Eclipse platform that allows to adopt different formats of data sources; as well as utilizing different forms of preprocessing, system analysis, and searching algorithms. This is done by implementing a standard Eclipse plug-in and installing it within the Eclipse plug-in directory, without modifying any source code of Alborz or violating the integrity of the user interface of Alborz or Eclipse. The components of Alborz are briefly discussed below.

User interface component: UI closely collaborates with and controls other components within the Alborz toolkit and interacts with the user throughout the *offline* and *online* analysis phases. Almost all the events and requests in the system are emitted from this component. UI consists of the following parts:

- *System data generation wizard*: this wizard controls the operations of the *I/O component* and *preprocessing component* by guiding the offline operations within the toolkit. It allows the user to: i) extract data from various sources (source code, GXL, RSF, etc.) for static analysis; ii) extract execution traces for dynamic analysis; iii) perform data-mining operations to extract association relation among system entities; iv) generate sequential patterns from execution traces; v) generate system data including domains of entities and similarity matrices for the A* search algorithm; and vi) store system data in the local Datastore to be used for the online system analysis.
- *System analysis wizard*: this wizard assists the user throughout the steps of the online system analysis, such as: analysis-type selection; main seed selection; query editing; and redistribution of entities into components.
- *Perspectives, views, menus and tool-bars*: a series of standard Eclipse user-interface elements used for integrating Alborz with the Eclipse platform.
- *Command line interpreter*: with basic editing facilities to directly launch specific operations.

Datastore component: Datastore is the center of the Alborz structure and allows the system components to communicate with each other through this component's interfaces. XML files are used for storing: source graphs; domains of entities for the search engine; similarity matrices; query forms; result of architectural analysis; execution traces and patterns; and annotated source code that allows navigating the source code from the solution of the recovery.

System analysis component: encapsulates all the functionalities that are required in the online analysis phase, including: selecting main-seeds as the core functionality in each component; executing a sub-optimal version of the A* search algorithm; and performing dynamic analysis. To fulfill tool extensibility, we utilized design patterns in the design of system analysis component and support Eclipse plug-ins to add other search algorithms and analysis types (e.g., clustering).

I/O component: provides interoperability among the systems in the environment of Alborz and the Alborz toolkit. It receives inputs such as: source code, GXL, RSF (for static analysis) and execution traces (for dynamic analysis). It also uses the *Evaluation component* to evaluate the recovery results, as its output. I/O component supports plug-in extensibility and design patterns as well. Furthermore, this component provides parser wrappers to wrap external parsers for parsing source code of the subject software system, and provides a wrapper for the SHriMP visualization tool to connect it to Alborz.

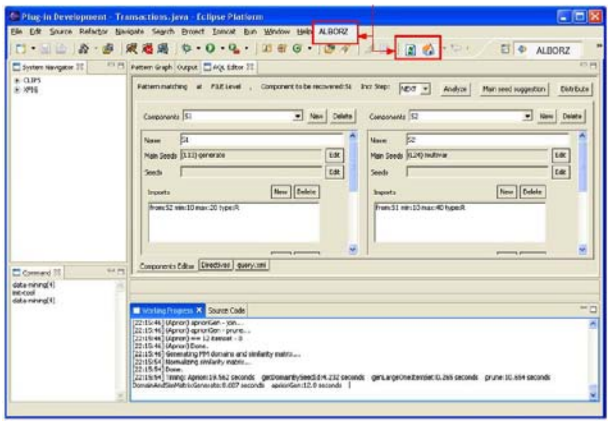
5. Example Tool Usage

In this section, the candidate experimentations to extract the static and dynamic views of the Xfig drawing tool are briefly discussed. For the static view in Figure 2, a high-level structure decomposition of Xfig is specified in terms of subsystems and interconnections (at file-level); then each subsystem can be specified as a number of modules and interconnections (at function-level).

For the dynamic view, Table 1 left column presents a number of specific features from the related feature-specific scenario sets (namely feature-families) that are applied on the Xfig tool. The other columns of Table 1 represent a variety of statistics related to the extracted functions that implement these features.

Feature Family	Specific Feature of Xfig	Number of Different Scenarios	Average Trace Size	Average Pruned Trace Size	Number of Extracted Patterns	Average Pattern Size
Draw Ellipse	Circle-Diameter	10	7234	2600	46	33
	Circle-Radius	10	8143	2463	48	32
	Ellipse-Diameter	10	6405	2536	41	37
	Ellipse-Radius	10	7351	2549	39	35
Copy	Move Objects	4	11887	3166	31	53
	Copy Objects	4	11460	3269	37	50
Draw Spline	Closed Interpolated	10	18635	4434	58	63
	Interpolated	10	15469	4038	66	49
	Approximated	10	15057	5362	61	47

Table 1. The results of execution pattern mining for 3 Xfig feature families.


Plug-in Development - Transactions.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Tomcat Run Window Help ALBORZ
ALBORZ
System Navigator
CLIPS
XFIG
Pattern Graph Output AQL Editor
Analyze
Main seed suggestion
Distribute
New Delete Edit
Components Editor Directives query.xml
Command
data mining(4)
init-cool
data mining(4)
Working Progress Source Code

Pattern matching at FILE level , Component to be recovered:S1 Incr Step: NEXT

Components: S1
Name: S1
Main Seeds: (113) generate
Seeds:
Inputs: from:S2 min:10 max:20 type:R

Components: S2
Name: S2
Main Seeds: (124) multivar
Seeds:
Inputs: from:S1 min:10 max:40 type:R

```
[22:15:46] (Apriori) aprioriGen - join...
[22:15:46] (Apriori) aprioriGen - prune...
[22:15:46] (Apriori) === 12 itemset - 0
[22:15:46] (Apriori) Done.
[22:15:46] Generating FPM domains and similarity matrix...
[22:15:54] Normalizing similarity matrix...
[22:15:54] Done.
[22:15:54] Timing: Apriori:19.562 seconds   getDomainBySeed:14.232 seconds   genLargeOneItemset:0.265 seconds   prune:10.654 seconds
DomainAndSimMatrixGenerate:8.607 seconds   aprioriGen:12.0 seconds
```

Figure 2. Screen snapshot of Alborz toolkit

6. Limitations and Future Extensions

Alborz tool has some limitations as described below. In static analysis, due to the required computations to evaluate component-interaction constraints at both function level and file level, the size range of the subject software systems is limited to middle-size systems (approximately less than 300 KLOC). The current version of Alborz only covers procedural software systems and lacks a fact extractor tool; however, the tool interoperates with the existing fact extractor tools and accepts input data in a variety of formats (e.g., GXL, RSF). In dynamic analysis, the trace extraction is performed through execution of a set of task scenarios; hence, the familiarity of the user with the application domain and the subject system is required.

The future extensions to the tool include the provision of: i) static and dynamic analysis of object oriented systems; ii) integrated multi-view analysis, where the result of dynamic view is used to enhance the result of static view; and iii) a design view based on analysis of a repository of design decision facts.

7. Conclusion

In this paper, we introduced the new version of Alborz toolkit as a multi-view software architecture analysis tool that is developed as an Eclipse plug-in. The static view exploits a pattern matching mechanism that maps a high-level conceptual architecture of a software system onto the source graph of the system. The dynamic view allows to map the core functionality of software features onto the software's structure and to evaluate its structural quality.

References

[1] Eclipse: universal tool platform, URL = http://www.eclipse.org/.

[2] K. C. et al. DRT: a tool for design recovery of interactive graphical applications. In *Proceedings of the ICSE'03*, pages 814–815, 2003.

[3] A. Hamou-Lhadj and T. C. Lethbridge. A survey of trace exploration tools and techniques. In *CASCON'04*, pages 42–55, 2004.

[4] D. F. Jerding and S. Rugaber. Using visualization for architectural localization and extraction. *Science of Computer Programming*, 36(2-3):267–284, 2000.

[5] 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'98*, pages 45–53, 1998.

[6] Reasoning Systems Inc., Palo Alto, CA. *Refine User's Guide*, version 3.0 edition, 1990.

[7] K. Sartipi. Alborz: A query-based tool for software architecture recovery. In *Proceedings of the IWPC'01*, pages 115–116, Toronto, Canada, 2001.

[8] K. Sartipi and K. Kontogiannis. On modeling software architecture recovery as graph matching. In *Proceedings of the ICSM'03*, pages 224–234, 2003.

[9] C. Stoermer, L. O'Brien, and C. Verhoef. Practice patterns for architecture reconstruction. In *Proceedings of the WCRE'02*, pages 151–160, 2002.

[10] M.-A. Storey, C. Best, and J. Michaud. Shrimp views, an interactive environment for exploring java programs. In *Proceedings of the IWPC'01*, pages 111–112, 2001.