ArchViz: a Tool to Support Architecture Recovery Research

Vanius Zapalowski[1], **Ingrid Nunes**[1], **and Daltro José Nunes**[1]

[1]Prosoft Research Group – Instituto de Informática
Universidade Federal do Rio Grande do Sul, Brazil

{vzapalowski,ingridnunes,daltro}@inf.ufrgs.br

Abstract. *In order to produce documented software architectures, many software architecture recovery methods have been proposed. Developing such methods involves a not trivial data analysis, and this calls for different data visualisations, which help compare predicted and target software architectures. Moreover, comparing methods is also difficult, because they use divergent measurements to evaluate their performance. With the goal of improving and supporting architecture recovery research, we developed the ArchViz tool, which is presented in this paper. Our tool provides metrics and visualisations of software architectures, supporting the analysis of the output of architecture recovery methods, and possibly the standardisation of their evaluation and comparison.*
Video link: `http://youtu.be/Gjo5cOzk4kM`.

1. Introduction

An explicitly documented software architecture plays a key role in software development as it keeps track of many design decisions and helps maintain consistence in the developed software. It provides useful knowledge to deal with the software evolution accordingly to planned architectural principles captured by a high-level model — usually represented in a graphical model. Despite of the importance of having a documented architecture, many systems lack proper architectural documentation.

Software architecture recovery (SAR) methods aid software architects in the task of inspecting the source code to understand an implemented software when there is no architectural documentation available or it is outdated. SAR methods have been proposed to reduce the human effort needed to perform this task. Such methods use different inputs (e.g., dependencies, semantics, and patterns) and a variety of metrics (e.g., precision, recall, and distance) to produce recovered architectures. Moreover, SAR studies focus on the measurement of certain properties lacking a visual representation of their recovered and target architectures [Ducasse and Pollet 2009]. Consequently, the process of evaluating and analysing results of a SAR method is a complex and time-consuming activity [Garcia et al. 2013], given the combination of possible sources of information, evaluation metrics, and results analysis.

Different tools have been proposed in the literature to improve software architectures, e.g. [Lindvall and Muthig 2008], and most of them focus on checking architecture conformance or compliance. On the other hand, ArchViz, the tool introduced in this paper, has the goal of supporting SAR *research*. Therefore, the purpose of this new tool is the key difference from other existing tools in the context on software architecture. In previous work [Zapalowski et al. 2014], we faced the problems discussed above in a study to evaluate the relevance of code-based characteristics to identify modules of recovered architectures. To address such problems, we implemented a web-based tool, named **ArchViz**, able to partially automate the analysis of recovered architectures, as no other similar tool was available. Therefore, our tool emerged from our own (real) need for supporting our research, and its effectiveness is indicated by the research results we were able to derive from our data analysis with the support of ArchViz. Our tool provides *evaluation metrics* of recovered software architectures using well-known information retrieval measures. In addition, our tool generates *three visualisations* (tree-map, module dependencies graph and element dependencies graph) of recovered (or predicted) and target architectures in order to help understand the results of the recovery process.

This paper is organised as follows. Section 2 describes ArchViz, presenting its main features. Next, Section 3 discusses existing tools related to software architecture visualisation and their support to SAR. Finally, we present the final remarks in Section 4.

2. The ArchViz Tool

This section presents the contributions that ArchViz provides to support SAR research. Our intended users are researchers, which are able to compare a target architecture (oracle) with a recovered architecture, and verify metrics that indicate the classification effectiveness. First, we describe in Section 2.1 ArchViz's architecture and how to use it, presenting its user interface. In Section 2.3, we present the two main features of ArchViz: (i) measurement of well-known information retrieval metrics, which are adopted in the context of general-purpose classification problems, and are detailed in Section 2.2; and (ii) plotting of three different graphical models of recovered and target architectures.

In order to illustrate the functionalities of ArchViz, we present the evaluation and analysis of one of the five subject system used in our previous work [Zapalowski et al. 2014], named OLIS. Thus, the metrics and visualisations presented in the remainder of this paper are extracted from OLIS, which is an agent-based product line that provides personal services for users.

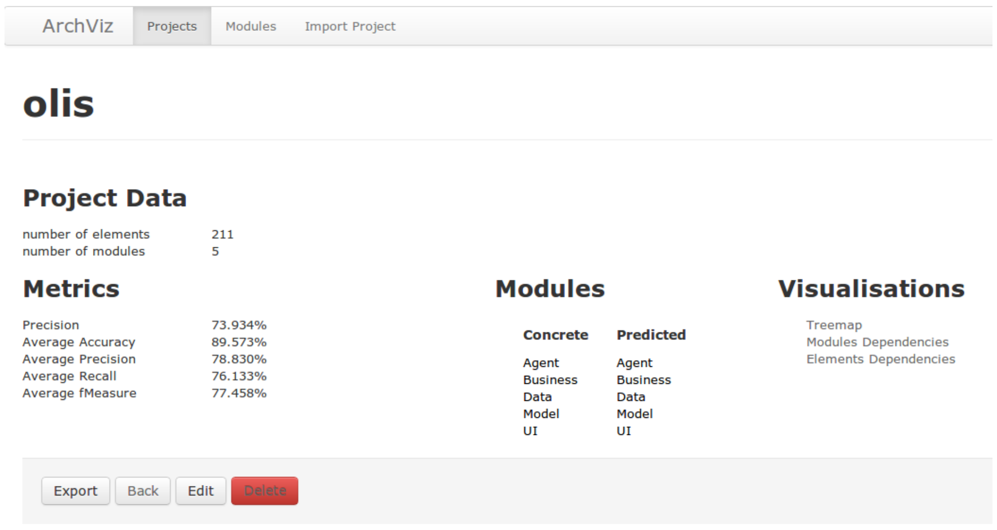
2.1. Architecture and User Interface

ArchViz is a web-based application implemented in Ruby using the Ruby on Rails (RoR)[1] framework. Consequently, the architecture of our tool follows the Model–View–Controller architectural pattern adopted by RoR. In our implementation, the main task of each architectural module are: the *Model* represents and stores imported architectures; the *Controller* calculates the implemented evaluation metrics; and the *View* plots the architectural visualisations using D3[2] JavaScript library.

To start using ArchViz, users should import a software project using the *Import Project* option available in the menu, which allows users to provide input data. Projects' details must be specified in two Comma-Separated Values (CSV) files: (i) the first with information about to which module each architectural element belongs in the recovered and in the target architectures; and (ii) the second with all the dependencies between architectural elements. The adequate format of such files is detailed in the functionality

[1]Available at `http://rubyonrails.org/`
[2]Available at `http://d3js.org`

of importing projects. After importing a project, our tool summarises and presents the data related to it, as illustrated in Figure 1.



olis

Project Data

number of elements	211
number of modules	5

Metrics

Precision	73.934%
Average Accuracy	89.573%
Average Precision	78.830%
Average Recall	76.133%
Average fMeasure	77.458%

Modules

Concrete	Predicted
Agent	Agent
Business	Business
Data	Data
Model	Model
UI	UI

Visualisations

Treemap
Modules Dependencies
Elements Dependencies

Figure 1. ArchViz User Interface.

2.2. Architecture Recovery Metrics

We selected general purpose metrics to evaluate multi-class prediction of machine learning algorithms, defined by Sokolova and Lapalme [Sokolova and Lapalme 2009], as the metrics to evaluate the quality of the recovered architecture, because they are also applicable to our context. Using these metrics, we are able to standardise the analysis of results of SAR methods, given that this set of metrics only needs the recovered and target architectures to be calculated. The metrics definitions are given in Table 1, following this notation: K is the set of proposed architectural modules; i is a module such that $i \in K$; $|K|$ represents the cardinality of K; tp_i are the true positives of i; tn_i are the true negatives of i; fp_i are the false positives of i; fn_i are the false negatives of i; and $|i|$ is the number of elements in the module i. The definition of what is an element is specific to each recovery method: it can be a class, a components, a procedure, and so on.

2.3. Architecture Visualizations

Because of the complexity of large-scale software, it is difficult to represent it in a single simple model. Software architecture visualisation helps stakeholders involved with software development to understand the concepts adopted in their applications using a high-level representation. Most of the SAR approaches focus on presenting metrics to evaluate their results, and they do not provide architectural visualisations that enable a finer-grained analysis of results. This is helpful particularly to researchers, because humans can derive findings based on visual models and data abstractions better than machines [Keim et al. 2008]. Therefore, we proposed and implemented three visualisations that aim to improve the analysis and comparison of recovered and target architectures.

We next present the three visualisations that our tool provides: (i) *Tree-map*, which provides a visual analysis of the recovered architecture using a hierarchical representation (Section 2.3.1); (ii) *Module Dependencies Graph*, which details dependencies

Table 1. Metrics implemented in ArchViz.

Metric	Description	Formula
Precision	*The precision measures the correctness of the overall recovered architecture independently from architectural modules size. It considers only the tp of each module.*	$\frac{\sum_{i=1}^{K} tp_i}{\sum_{i=1}^{K} \lvert i \rvert}$
Average Precision	*To evaluate the per-module precision, the average precision measures the agreement between the recovered and the target architecture for each module. It only considers the cases where the recovered classification of the architectural elements agrees with the target architecture.*	$\frac{\sum_{i=1}^{K} \frac{tp_i}{tp_i+fp_i}}{\lvert K \rvert}$
Average Recall	*By calculating the average recall, we obtain an average of the per-class effectiveness of an SAR method to identify architectural modules. To calculate the average recall, we consider the tp and fn of each module.*	$\frac{\sum_{i=1}^{K} \frac{tp_i}{tp_i+fn_i}}{\lvert K \rvert}$
Average Accuracy	*The average accuracy measures the correctness of each module and the distinctness from the other modules. It evaluates the correct architectural elements, tp and tn, of each recovered module. The average accuracy is useful to measure the recovery method per-module effectiveness.*	$\frac{\sum_{i=1}^{K} \frac{tp_i+tn_i}{tp_i+tn_i+fp_i+fn_i}}{\lvert K \rvert}$
Average F-measure	*The average F-measure combines the average precision and average recall to provide one metric that indicated both the overall correctness and the module prediction quality. This metric measures the relationship between correctly predicted elements and those given by a metric based on a per-module average.*	$\frac{2*avg\ prec*avg\ rec}{(avg\ prec+avg\ rec)}$

among modules, and their respective sizes (Section 2.3.2); and (iii) *Element Dependencies Graph*, which presents a fine-grained visualisation of architectural elements showing only the inter-modules dependencies (Section 2.3.3). Note that, in our tool, all visualisations are shown together with the metrics described previously. Moreover, the last two visualisation types show two graphs corresponding to the recovered and target architectures, allowing a side-by-side comparison.

2.3.1. Tree-map

The tree-map visualisation is a two-dimensional hierarchy graph created by Shneiderman [Shneiderman 1992] to analyse hard disk usage. Similarly to software architectures, the disk folder hierarchy represents categories, and files are leaf elements that are in a folder. A common problem to visualise the data is to represent the relevance of more than two attributes in a single chart. In the hard disk usage, for example, files have a parent folder and size, which means that we need to plot separate graphs for file usage and for folder usage, to visualise both attributes using a Cartesian coordinate system. Shneiderman proposed a tree-organisation structure, where each element is represented as a rectangle and attributes can be specified by colours, sizes or hierarchy position of the rectangles. Then, the hard disk usage can be represented in a single graph, where folders are outside rectangles with its file elements inside. Additionally, their sizes can represent the amount of disk usage.

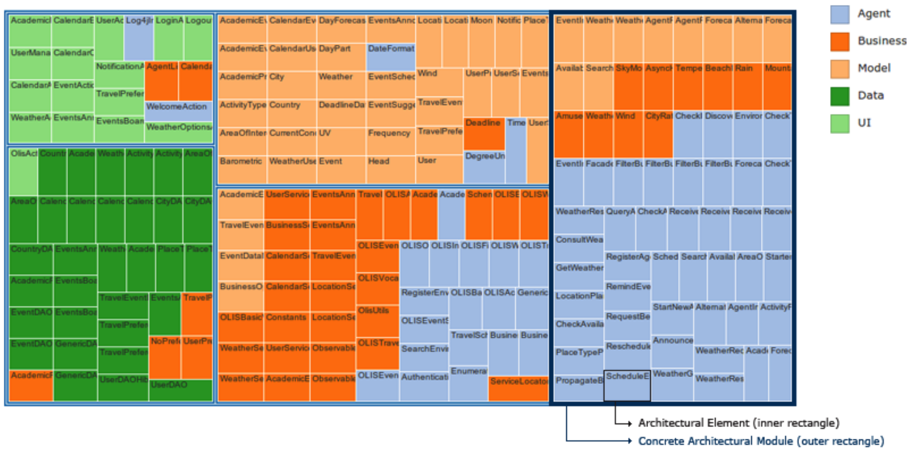


Figure 2. Tree-map Visualisation of the OLIS Recovered Architecture.

As in hard disk usage, a software architecture typically has a hierarchical structure: architectural elements belong to modules. So, we mapped software architectures to the tree-map representation, in order to understand the predicted results of the recovered and target architectures. We represent both architecture versions in a single graph to visualise predicted results. Figure 2 is an example of the tree-map visualisation, where the outer rectangles are the target architectural modules, the inner rectangles are the architectural elements with their name, and colours of architectural elements are assigned according to the recovered module to which they belong. The target module names, shown in the upper right hand side of Figure 2, are possibly from a manually recovered architecture. In the case when the recovered architecture matches the target architecture completely, all outer rectangles are coloured by only one colour and each outer rectangle has a different colour from the others. Figure 2 illustrates a scenario in which the recovered architecture differs from the target architecture. As can be seen in Figure 2, the outer rectangles major colour defines its recovered architectural module, i.e. in Figure 2, the lower left rectangle corresponds to the Data module and the upper left rectangle corresponds to the UI module.

Figure 2 thus indicates the recovered modules and the assignment distribution of architectural elements to the target modules. Furthermore, this representation confirms the information provided by the module accuracy: (i) if a single colour is concentrated in a single module, accuracy is high; and (ii) otherwise, it is low. Additionally, the tree-map visualisation combines the recovered and target architecture allowing a visual comparison of the architectural measures extracted from a SAR method.

2.3.2. Module Dependencies Graph

The module dependencies graph is a coarse-grained view that aims to provide an overall view of the system. It is similar to the the most common notation to represent architectures used, where architectural modules are represented as nodes and communication among them as edges. This representation improves architecture understanding, because it exposes the main system concepts, and presents them in a concise way, showing both the architectural modules and how they communicate to each other. Figure 3 shows an example of this notation, presenting the architecture of OLIS, which uses a layered architectural pattern.

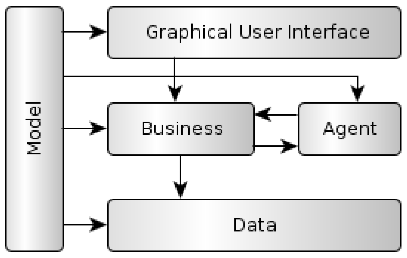


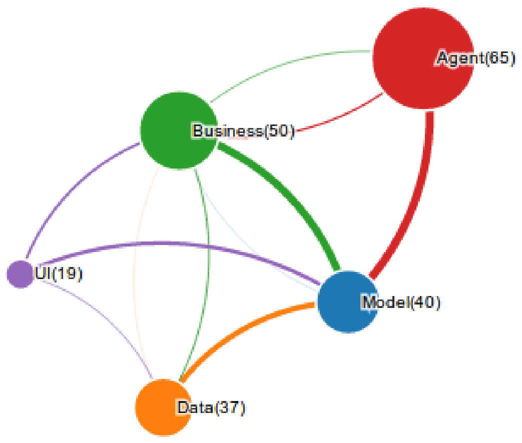
Figure 3. Example of a Typical Architecture Model.

Although this traditional model presents, in a high-level view, the main architectural modules and their communication, it lacks important details needed to compare the recovered and target architectures. Furthermore, it undertakes architectural information that could be represented in an architectural visualisation, such as the intensity of the dependency among two modules, which is helpful for understanding a recovered architecture. Analysing the representation of the OLIS architecture in Figure 3, it is impossible to identify the intensity of dependencies among modules. In this usual representation, the module sizes represent just the existence of architectural modules and they do not correspond to the size of the modules in the system.

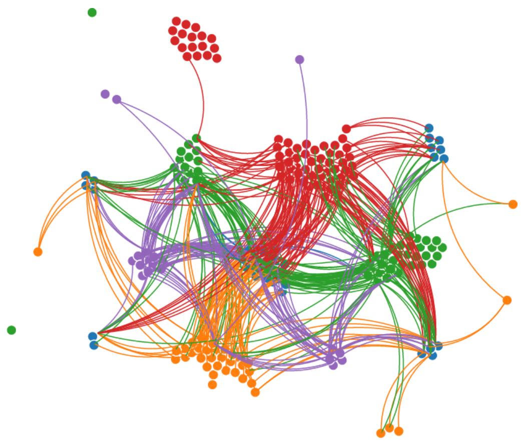
To enrich the information provided by this traditional module dependencies graph, we implemented the module visualisation with modifications. The same architecture presented in Figure 3 is represented in ArchViz as shown in Figure 4(a). In ArchViz, the modules are defined by their size and colour. Their colours characterise each module role and their sizes are proportional to the number of architectural elements that they have. Additionally, we add labels to the modules nodes with the architectural role and number of elements that they have. The edges represent the communication among modules specifying the dependency hierarchy, i.e. an edge in red means that the red module uses the module that it is linked to. Moreover, the edge thickness is proportional to the dependency level that the modules has, e.g. the dependency among the Agent and Model modules is stronger than that of the Agent and Business modules, in the presented OLIS architecture.

2.3.3. Element Dependencies Graph

The element dependencies graph is the finest-grained architectural visualisation that ArchViz provides. It presents the dependencies among architectural elements classifying them into architectural modules. This visualisation represents elements as nodes, and the node colours characterise to which module they belong. The edges are coloured by the inter-module dependency, similarly to the module dependencies graph. This representation disregards the intra-module dependencies to reduce the number shown edges — otherwise the graph would provide too much information. As an example, we present the OLIS target architecture using the element dependencies graph in Figure 4(b). The graph



(a) Module Dependencies.



(b) Element Dependencies.

Figure 4. OLIS Graphs.

shows: (i) five modules, (ii) the inter-modules dependencies, and (iii) the 211 elements of the system.

3. Related Work

Besides previous work we already discussed, which defines adopted metrics and inspired our visualisations, there are important studies that are closely related to ArchViz. We discuss these studies in this section.

A metric that is commonly used to evaluate recovered architectures is the MoJo distance metric [Tzerpos and Holt 1999]. It is a domain-specific metric to measure the number of steps needed to obtain the target architecture, given a recovered architecture. It was not implemented in ArchViz due to problems on its application in some specific SAR methods that were recently reported [Garcia et al. 2013].

Bunch [Mancoridis et al. 1999] is one of the first tools that support all the software decomposition process, from the manual investigation to the recovered architecture visualisation. It generates subsystems and creates a fine-grained representation of the software based on the architectural element dependencies, similar to that presented in Figure 3. Differently from ArchViz, Bunch's objective is to obtain a recovered architecture. Thus, it does not comprise the evaluation metrics and comparisons of the recovered against the target architecture.

An approach that compares architectural models was proposed by Beck and Diehl [Beck and Diehl 2010]. Their approach evaluates the similarity of architectural element dependencies using a matrix dependencies representation. This method points out similarities and divergences of the architectures in the architectural elements level. Therefore, it analyses only the similarities in the elements, and modules information, such as modules communication, are not taken into account.

4. Conclusion

Software architecture recovery (SAR) methods have been proposed to decrease the effort needed to maintain up-to-date architectural documentation of software systems. These methods apply different evaluation metrics to analyse recovered architectures and to be used as basis to derive their findings. In addition, SAR methods often lack a visual representation of their recovered and target architectures, which are essential to analyse results.

We built ArchViz to address these issues to support SAR research, by providing measurement of evaluation metrics and architecture visualisation representations. The implemented metrics provide statistical evidences of the level of agreement between recovered and target architectures. Moreover, we provided visualisations and side-by-side comparisons of recovered and concrete architectures contribute with useful knowledge to understand results of a method, which helps in the process of refining and improving it. Thus, ArchViz is a tool that reduces the effort needed to analyse the recovered architectures, providing a useful set of metrics together with an automatic generation of architectural models to support the SAR research.

It is important to highlight that one of the subject systems investigated in our research on SAR using ArchViz is from the industry. However, this system was not presented in this paper (but the OLIS) due to a confidentiality agreement. Although we used the tool in a real world scenario, we have not used it with (very) large scale systems, and this is part of our future work.

References

[Beck and Diehl 2010] Beck, F. and Diehl, S. (2010). Visual comparison of software architectures. In *International Symposium on Software Visualization*, pages 183–192.

[Ducasse and Pollet 2009] Ducasse, S. and Pollet, D. (2009). Software architecture reconstruction: A process-oriented taxonomy. *Trans. Softw. Eng.*, pages 573–591.

[Garcia et al. 2013] Garcia, J., Ivkovic, I., and Medvidovic, N. (2013). A comparative analysis of software architecture recovery techniques. In *International Conference on Automated Software Engineering*, pages 486–496.

[Keim et al. 2008] Keim, D., Mansmann, F., Schneidewind, J., Thomas, J., and Ziegler, H. (2008). Visual analytics: Scope and challenges. In *Visual Data Mining*, pages 76–90.

[Lindvall and Muthig 2008] Lindvall, M. and Muthig, D. (2008). Bridging the software architecture gap. *Computer*, 41(6):98–101.

[Mancoridis et al. 1999] Mancoridis, S., Mitchell, B. S., Chen, Y., and Gansner, E. R. (1999). Bunch: A clustering tool for the recovery and maintenance of software system structures. In *International Conference on Software Maintenance*, pages 50–59.

[Shneiderman 1992] Shneiderman, B. (1992). Tree visualization with tree-maps: 2-d space-filling approach. *Transactions Graphics*, pages 92–99.

[Sokolova and Lapalme 2009] Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Inform. Process. Manag.*, pages 427–437.

[Tzerpos and Holt 1999] Tzerpos, V. and Holt, R. C. (1999). Mojo: A distance metric for software clusterings. In *Working Conference on Reverse Engineering*, pages 187–193.

[Zapalowski et al. 2014] Zapalowski, V., Nunes, I., and Nunes, D. (2014). Revealing the relationship between architectural elements and source code characteristics. In *International Conference on Program Comprehension*, pages 14–25.