Abstracting Module Views from Source Code *

Martin Pinzger, Michael Fischer, Mehdi Jazayeri
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1
A-1040 Vienna, Austria
{pinzger, fischer, jazayeri}@infosys.tuwien.ac.at

Harald Gall
Department of Informatics
University of Zurich
Winterthurerstrasse 190
CH-8057 Zurich, Switzerland
gall@ifi.unizh.ch

Abstract

In this paper we present ArchView an approach for abstracting and visualizing software module views from source code. ArchView computes abstraction metrics that are used to filter out architectural elements and relationships of minor interest resulting in more reasonable and comprehensible module views on software architectures.

1. Motivation

Views on the architecture of software systems aid engineers in maintaining and evolving software systems. However, abstracted views as obtained by existing reverse engineering techniques often are too low level and cluttered with information making them almost useless.

In this paper we address this problem and introduce ArchView a view abstraction approach that concentrates on the abstraction and visualization of *condensed* architectural views. ArchView is based on the abstraction approach of Holt et al. described in [2]. A similar approach also has been presented by Feijs et al. in [1]. Our approach uses the techniques of both approaches but extends them by focusing on the computation of abstraction metrics. They then are used to highlight interesting architectural elements and relationships and filter information of minor interest.

2. ArchView Approach

Abstraction: The abstraction approach of ArchView uses binary realtional algebra to abstract/lift information along a containement hierarchy as given by the underlying source code meta model. For example, a software module is implemented by package *A* which contains a number of sub-packages. Each package contains a set of classes that further contain methods and attributes. Using these *contains* relationships the algorithm first determines the set of entities (e.g. methods) contained by each module and next computes the relationships (e.g. invokes) between these entities. Whenever a relationship between the entities of module *A* and module *B* occurs an abstracted relationship of this type is established between modulel *A* and *B*.

Visualization: The visualization technique used by ArchView is an extension of the technique presented by Lanza et al. in [3]. The extension is concerned with showing the weights of relationships and filtering mechanisms. Filtering is based on the abstraction metrics computed during the abstraction process. For instance, thresholds are used to filter minor entities and relationships. This yields to more condensed architecural views.

Case Study and Results: To demonstrate and validate the ArchView approach we applied it to the open source web browser Mozilla. The focus of the case study was to analyze source code related dependencies between a selected set of Mozilla's sofware modules. By computing the abstraction metrics and applying filtering techniques more condensed views on selected software modules have been retrieved than by using the existing techniques.

*This work is partially funded by the Austrian Forschungsförderungsfonds für die Gewerbliche Wirtschaft (FFF) and the European Commission under EUREKA 2023/ITEA-ip02009 'FAct-based Maturity through Institutionalisation Lessons-learned and Involved Exploration of System-family engineering (FAMILIES)'.

References

[1] L. Feijs, R. Krikhaar, and R. van Ommering. A relational approach to support software architecture analysis. *Software Practice and Experience*, 28(4):371–400, 1998.

[2] R. C. Holt. Structural manipulations of software architecture using tarski relational algebra. In *Proceedings of the Working Conference on Reverse Engineering*, pages 210–219. IEEE Computer Society Press, 1998.

[3] M. Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *Proceedings of IWPSE 2001 (International Workshop on Principles of Software Evolution)*, 2001.