Model-Based Architecture Restructuring of a Large Industrial System

Niels Streekmann
OFFIS - Institute for Information Technology
Technology Cluster Software Engineering and Enterprise Architecture
niels.streekmann@offis.de

Abstract

In current practice architecture restructuring requires a high manual effort for understanding the existing system and the defintion of a mapping of its elements to a target architecture. Our model-based architecture restructuring approach supports reengineers by semi-automating this task. This paper presents the first results of an industrial case study with a large productive system.

1 Model-Based Architecture Restructuring

Our approach to model-based architecture restructuring is based on a structural model of the existing system and a model of the target architecture. The connection between the two models is created by an initial mapping model which maps a small part of the structural elements of the system (e.g. files, classes and methods) to components of the target architecture.

Based on the initial mapping a graph clustering algorithm is used to create a complete mapping of all structural elements to target components. The algorithm implements a variant of hierarchical clustering which considers the target architecture in its termination condition. The nodes of the graph are the structural elements of the system while the edges represent the dependencies between these elements. The edges are weighted according to the types of the dependencies. The distance function used for the hierarchical clustering is the sum of these weights.

The approach is iterative in order to improve the clustering result by changing the initial mapping, dependency weights or the target architecture. Furthermore, iterations can be used to allow the stepwise restructuring of the system in horizontal or vertical partitions. A more detailed description of the approach can be found in [2].

2 Evaluation

2.1 Metrics

Since the evaluation context is an ongoing industry project, the optimal result of the clustering is not known. As a consequence, metrics such as the number of correctly assigned source elements can not be used to rate the clustering result. Applicable metrics e.g. are the number of nodes per cluster and the rating by system experts.

Furthermore, best practices for the definition of dependency weights and the composition of the initial mapping are still to be evaluated. Therefore, another metric that turned out well is the difference between two clustering runs with different inputs (for e.g. dependency weights or the initial mapping). The inspection of the nodes that moved to different clusters from one run to another gave valueable insights and was easier to handle than the evaluation of a single clustering result.

2.2 System Under Study

The system under study is a productive system that has been maintained for 15 years. It is written in C/C++ and contains approx. 4 million lines of code in 4915 files. The system is partly object-oriented and much of the communication and synchronisation in the system is done using 3940 global variables and 361 shared memory objects.

Between the structural elements there are 632300 dependencies. The types of dependencies and their frequency are listed in Table 1. The numbers in the table are summed up from more fine-grained dependency types, where some of these types are counted for more than one subsumed dependency type. Therefore the sum of the numbers in the table exceeds 632300. The weights are explained in section 2.3.

The longtime maintenance of the system led to architecture erosion. Hence, plans for future extensions are not efficiently feasible on the current implementation. This is the reason for a restructuring of the system following functional viewpoints.

The target architecture does not use components as e.g. defined by [3], but is built using the rules for software categories described in [1]. It consists of 13 categories with 20 subcategories.

2.3 Experiment Setting

For performance reasons, the nodes of the input graph for the clustering do not comprise all structural elements of the source system. Only files, global variables and shared memory objects are considered as nodes for the clustering, which sums up to 9216 nodes. The dependencies of the structural elements in the

Dependency Type	Number of Dependencies	Weight
Function Execution	243972	0,01
Read Access	141685	0,003
Macro Access	80713	0,00000001
Write Access	45950	20
Variable Type Usage	45491	0,00000001
File Inclusion	44829	0,01
Inheritance	19322	200
Containment	17619	1,0
Object Member Access	17583	0,1
Class Friendship	92	100

Table 1: Dependency Types

files are lifted to the file level. I.e. dependencies inside files are not considered and the files themselves are not restructured.

Furthermore, dependencies are subsumed in dependency objects which represent all dependencies of a certain type between two nodes. This reduces the 632300 dependencies to 123330 dependency objects. The dependencies are weighted in order to indicate their relevance for the clustering, i.e. their indication for the functional similarity of the nodes that are connected by the dependency. The weights of the different dependency types (c.f. Table 1) are based on experiences with previous clusterings. E.g. types that indicate high functional cohesion (e.g. Inheritance and Write Access) are given high weights.

2.4 Results

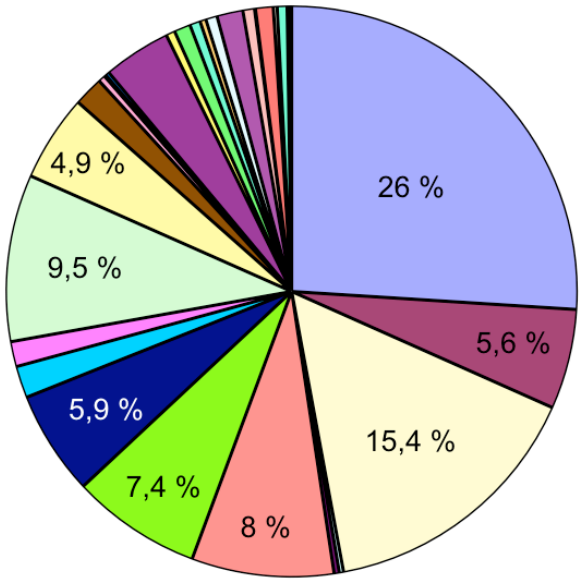

Figure 1: Clustering Result

Figure 1 shows a current clustering result. Each component and subcomponent of the target architecture is represented by a separate cluster. It can be seen, that the largest cluster contains 26% of the nodes. The smallest cluster does only contain 2 nodes that were assigned in the initial mapping. Thus, the clustering does not result in a homogenous distribution of nodes to clusters. This distribution however is in most cases reasonable considering the target architecture.

Since there is no complete mapping that can serve as a reference for the clustering, the results can not be validated completely. Thus, there are indications for correctly mapped nodes, but also some cases where the mapping is unexpected. In general one can say that the clustering provides useful information for the participating system experts.

The quality of the clustering can be determined by these experts by checking the correct mapping for single nodes or by evaluating the dependencies between clusters. To support this the dependencies can e.g. be classified into allowed, tolerated and forbidden dependencies.

The restructuring of the system under study is ongoing work, so the results presented here are not final. The final results can be used to determine the detailed target architecture of the restructured system and can be seen as a basis for the restructuring of the implementation of the system in order to implement the components of the target architecture.

3 Conclusions

This contribution details how the approach for model-based architecture restructuring presented in [2] can be applied to a large industrial system. It shows that the approach can be applied to such systems and provides useful results for system experts.

The restructuring of the industrial system will go on and therefore further insight into the quality of the approach and its applicability is expected. We further plan to evaluate the approach on a smaller open source system for which the optimal clustering is well known. This allows for further metrics on the clustering results and a sensitivity analysis of the algorithm since the analysis will be much more performant due to the size of the models.

References

[1] Johannes Siedersleben. *Moderne Softwarearchitektur*. Dpunkt Verlag, 2004.

[2] Niels Streekmann and Wilhelm Hasselbring. Model-Based Architecture Restructuring Using Graph Clustering. In *Workshop Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, pages 75–82, 2009.

[3] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2 edition, 2002.