Using Hypergraph Clustering for Software Architecture Reconstruction of Data-Tier Software

Ersin Ersoy[1], Kamer Kaya[2], Metin Altınışık,[1], and Hasan Sözer[3]

[1] Turkcell, Istanbul, Turkey
{ersin.ersoy,metin.altinisik}@turkcell.com.tr
[2] Sabanci University, Istanbul, Turkey
kaya@sabanciuniv.edu
[3] Ozyegin University, Istanbul, Turkey
hasan.sozer@ozyegin.edu.tr

Abstract. Software architecture reconstruction techniques aim at recovering software architecture documentation regarding a software system. These techniques mainly analyze coupling/dependencies among the software modules to group them and reason about the high-level structure of the system. Hereby, inter-dependencies among the software modules are mainly represented with design structure matrices or regular directed/undirected graphs. In this paper, we introduce a software architecture reconstruction approach that utilizes hypergraphs for representing inter-module dependencies. We focus on PL/SQL programs that are developed as data access tiers of business software. These programs are mainly composed of procedures that are coupled due to commonly accessed database elements. Hypergraphs are more appropriate for capturing this type of coupling, where an element can relate to more than one procedure. We illustrate the application of the approach with an industrial PL/SQL program from the telecommunications domain. We analyze and represent dependencies among the modules of this program in the form of a hypergraph. Then, we perform modularity clustering on this model and propose a packaging structure to the designer accordingly. We observed promising results in comparison with previous work. The accuracy of the results were also approved by domain experts.

Keywords: software architecture reconstruction; reverse engineering; hypergraph partitioning; modularity clustering; industrial case study.

1 Introduction

Modularity is one of the key properties for software design [16]. Especially large scale software systems need to have a modular structure. Otherwise, the maintainability and evolvability of the system suffer. A modular structure can be attained by decomposing the system into cohesive units that are loosely coupled. Software architecture design [3, 22] defines the gross-level decomposition

of a software system. Hence, its documentation is an important asset for coping with evolution [15].

Software architecture documentation might be incorrect/incomplete for old legacy systems due to *architectural drift* [14, 17]. Software architecture reconstruction (SAR) techniques [8] have been introduced to recover such documentation. These techniques mainly analyze coupling/dependencies among the software modules to group them and reason about the high-level structure of the system. Hereby, inter-dependencies among the software modules are mainly represented with design structure matrices (DSM) [2] or regular directed/undirected graphs [8]. These models capture direct dependencies between a pair of modules like call relations [13].

In this paper, we focus on PL/SQL programs that are developed as data access tiers of business software. These programs are mainly composed of procedures that are coupled due to commonly accessed database elements [2]. Existing SAR techniques do not consider indirect coupling/dependencies among the software modules based on such persistent data. Several procedures can be coupled due to a commonly accessed element. This type of coupling cannot be directly captured by existing models. Therefore, we introduce a SAR approach that uses a hypergraph model for representing such coupling/dependencies among modules. This model is partitioned to find clusters that maximize modularity. A packaging structure that is aligned with the obtained clusters is proposed to the designer. We illustrate the application of the approach with an industrial PL/SQL program from the telecommunications domain. We observed promising results with this case study in comparison with our previous work [2]. The accuracy of the results were also approved by domain experts.

The paper is organized as follows. In the following section, we provide background information on PL/SQL programs, hypergraphs and modularity clustering. We summarize the related studies in Section 3. We present the overall approach in Section 4. The approach is evaluated in Section 5, in the context of the industrial case study. Finally, in Section 6, we conclude the paper.

2 Background

2.1 PL/SQL Programs

PL/SQL (Procedural Language/Structured Query Language) combines procedural language features with the Structural Query Language (SQL) [1]. PL/SQL programs work on Oracle[4] database management system and they constitute significant part of enterprise applications today.

A PL/SQL program includes procedures that can be grouped into packages or remain as standalone procedures [2]. A sample PL/SQL procedure is shown in Listing 1.1. The first part of the procedure (Lines 1-4) declares variables and constants used in the application logic. The second part (Lines 5-19) contains

[4] www.oracle.com

the application logic. This part optionally includes a specification of exception conditions and their handling (Lines 13-18).

Listing 1.1 illustrates the interleaving of imperative code with SQL statements. Procedures are highly coupled with database elements and they are dependent on each other due to commonly accessed elements. In this work, we employ hypergraphs for representing these inter-dependencies. In the following, we shortly introduce the hypergraph formalism and our modeling approach.

Listing 1.1. A sample PL/SQL procedure [2].

```
1  PROCEDURE P(id IN NUMBER) IS
2   sales NUMBER;
3   total NUMBER;
4   ratio NUMBER;
5  BEGIN
6   SELECT x,y INTO sales,total
7    FROM result WHERE result_id = id;
8   ratio := sales/total;
9   IF ratio > 10 THEN
10   INSERT INTO comp VALUES (id,ratio);
11  END IF;
12  COMMIT;
13 EXCEPTION
14  WHEN ZERO_DIVIDE THEN
15   INSERT INTO comp VALUES (id,0);
16   COMMIT;
17  WHEN OTHERS THEN
18   ROLLBACK;
19 END;
```

2.2 Hypergraphs

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices. A net $n \in \mathcal{N}$ is a subset of vertices and the vertices in n are called its *pins*. The number of pins of a net is called the *size* of it, and the *degree* of a vertex is equal to the number of nets it belongs to. We use $\mathsf{pins}[n]$ and $\mathsf{nets}[v]$ to represent the pin set of a net n, and the set of nets containing a vertex v, respectively. In this work, we assume unit weights for all nets and vertices.

A K-*way partition* of a hypergraph $\mathcal{H}$ is a partition of its vertex set, which is denoted as $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$, where

- parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for all $1 \leq k < \ell \leq K$,
- each $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, i.e., $\mathcal{V}_k \subseteq \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$,
- the union of K parts is equal to $\mathcal{V}$, i.e., $\bigcup_{k=1}^{K} \mathcal{V}_k = \mathcal{V}$.

In our modeling approach, we represent each PL/SQL procedure as a vertex and each database table as a net. A net has several vertices as its pins if the corresponding procedures access the database table represented by the net. We convert this model to a weighted graph model and apply modularity clustering as explained in the following subsection.

2.3 Modularity Clustering

Given a (weighted) graph G, a good clustering of the vertices in G should contain G's edges within the clusters. However, since the number of clusters is not fixed, this objective can be trivially realized by a clustering that consists of a single cluster. Hence, alone, this objective is not a suitable clustering index. By adding a penalty term for larger clusters, we obtain the *modularity* of a clustering $\mathcal{C}$ [6]:

$$Q(\mathcal{C}) = \frac{\sum_{C_i \in \mathcal{C}} \omega(C_i)}{\omega(E)} - \frac{\sum_{C_i \in \mathcal{C}} (2 \times \omega(C_i) + cut(C_i))^2}{\alpha \times \omega(E)^2} \quad (1)$$

where $\omega(E)$ is the total edge weight in the graph, C_i is the i^{th} cluster, $\omega(C_i)$ is the total weight of internal edges in C_i, and $cut(C_i)$ is the total weight of the edges from the vertices in C_i to the vertices not in C_i. Like other clustering indices, modularity captures the inherent trade-off between increasing the number of clusters and keeping the size of the cuts between clusters small. Almost all clustering indices require algorithms to face such a trade-off. Hereby, α is a trade-off parameter, which determines the relative importance of the two trade-off dimensions. The value 4 is commonly assigned for α to establish equal/balanced importance. For this study, we have experimented with a range of α values and obtained the best results when α is equal to 2.8. We observed that the resulting number of clusters is aligned with the number of conceptual entities in the database. Hence, α can be adjusted based on a preprocessing of these entities. However, we left the automated adjustment of α parameter as future work.

3 Related Work

There exist many techniques [8] for SAR. Several of them use DSM for reasoning about architectural dependencies [2, 18–20] . Some focus on analyzing the runtime behavior for reconstructing execution scenarios [4] and behavioral views [12]. There are also tools that construct both structural and behavioral views [10,21] which are mainly developed for reverse engineering C/C++ or Java programs. Some tools are language independent; they take abstract inputs like module dependency graphs [13] or execution traces [4]. However, hypergraphs have not been utilized for SAR to the best of our knowledge.

There exist only a few studies [7,11] that focus on reverse engineering PL/SQL programs. They mainly aim at deriving business rules [7] and data flow graphs [11]. Recently, we proposed an approach for clustering PL/SQL procedures [2]. The actual coupling among these procedures can only be revealed based on their dependencies on database elements. In our previous work, we employed DSM [9] for representing these dependencies. In this work, we employ hypergraphs, which can more naturally model such dependencies and lead to more accurate results.

4 Software Reconstruction with Hypergraphs

The overall approach involves 4 steps as shown in Figure 1. First, the program source code and the database structure (meta-data) is provided to our *Dependency analyzer* tool as input (1). This tool creates a hypergraph model that represents dependencies among the procedures based on database tables that are commonly accessed. Second, the generated model is converted to a weighted graph (2). Then, this graph is recursively bi-partitioned by a clustering tool (3). Finally, the identified partitions are processed by our tool *Partition analyzer* to propose a package structure for the analyzed source code (4).

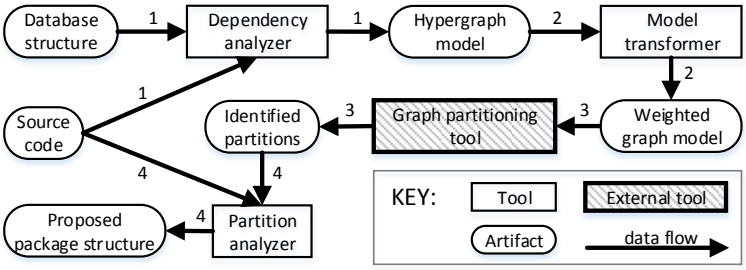

Fig. 1. The overall approach.

Dependency analyzer creates a hypergraph, where the number of vertices is equal to the number of procedures. Then, for each table in the database, it identifies the set of procedures that accesses the table. This set forms the set of pins for the net that represents the table.

To apply the modularity-based clustering, we transform the hypergraph into a weighted graph G as follows: each vertex in the hypergraph is also vertex of G and vice versa. Furthermore, there is an edge between two vertices u and v in the graph if they are connected via at least one net in the hypergraph. The weight of this (u, v) edge is assigned as $|\mathsf{nets}[u] \cap \mathsf{nets}[v]|$. After generating the weighted graph, we used the clustering tool by Çatalyürek et al. [5] to maximize the modularity. Starting with a single cluster G, the tool recursively partitions the clusters into two if the partitioning increases the modularity. We employ PaToH[5] as the inner partitioner in the clustering tool. In the following section, we illustrate the application of the approach in the context of an industrial case study from the telecommunications domain.

[5] http://bmi.osu.edu/umit/software.html

Table 1. A sample list of nets and the set of vertices they interconnect (pins) in the generated hypergraph for the CRM case study.

Net	*Vertices*
T1	*P119,P101,P1,P47,P15,P48*
T2	*P119,P57,P47,P26,P1*
...	*...*
T11	*P27,P26,P7,P1,P117,P119,P115,P111,P110,P109,...*
...	*...*
T67	*P8*

5 Industrial Case Study

We have performed a case study for automatically clustering modules of a legacy application implemented with the PL/SQL language. The application is a Customer Relation Management (CRM) system, which is maintained by Turkcell[6]. Its code comprises around 2 MLOC and the system is operational since 1993, serving more than 10000 users. In this section, we illustrate our approach for this system and discuss the results. We can not share real procedure/table names due to confidentiality; we present abstracted artifacts and results instead.

In our case study, we focused on one of the main schemas of the CRM system, which consists of 157 stored procedures and 690 tables. The same subject system[7] was used for evaluating our previous SAR approach [2]. We filtered out stored procedures that do not use any table. This preprocessing resulted with the final dataset that consists of 67 tables and 120 procedures. Hence, the created hypergraph has 120 vertices and 67 nets. Some of the nets are listed in Table 1 as an example. This hypergraph is processed as explained in Section 4 to derive a package structure for the procedures.

Results and Discussion: In total 9 partitions were obtained as listed in Table 2. Hereby, the number of items represent the number of procedures that are placed in the same partition. For instance, *Partition 3* includes 30 procedures. These procedures were not belonging to any package in the original application. They were defined as standalone procedures although they were working on the same database tables. We have validated this result with 4 different domain experts, all of whom agreed that these procedures perform related tasks and they should have been placed in the same package. We also observed that each partition can be mapped to a particular entity such as Customer, Address, Product etc. in the conceptual entity relationship model of the CRM database. The results regarding the partitions $5, 6, 7$ and 8 were also validated likewise. The validity of the other partitions $0, 1, 2$ and 4 were not confirmed by all the experts and they

[6] http://www.turkcell.com.tr

[7] The number of procedures and tables are slightly different compared to the previous study [2] due to the evolution of the system.

are also subject to some conflicts with respect to the conceptual entity relationship model. Finally, we compared these results with the results that we obtained using our previous approach [2] based on DSM clustering[8]. Hypergraph partitioning based approach turned out to be 20% better in terms of the percentage of procedures that are confirmed to be clustered correctly in a package.

Table 2. The set of partitions obtained as a result of clustering.

Partition	*# of items*	*Partition*	*# of items*	*Partition*	*# of items*
Partition 0	15	Partition 3	30	Partition 6	9
Partition 1	18	Partition 4	4	Partition 7	17
Partition 2	8	Partition 5	10	Partition 8	9
In total 9 partitions and 120 items					

There are several validity threats to our evaluation. First, the evaluation is based on subjective expert opinion rather than quantitative measurements. We tried to mitigate this threat by consulting 4 different experts and comparing the results with respect to their consistency with the conceptual entity relationship model. A second threat is regarding the use of a single subject system for the case study. Therefore, we plan to perform more case studies in the future. Although we focused on PL/SQL programs, our approach is relevant and applicable for any type of program that is highly coupled with a database management system.

6 Conclusion and Future Work

We introduced a software architecture reconstruction approach that employs hypergraph partitioning. We showed that hypergraphs can naturally represent dependencies that involve several modules. As a case study, we applied our approach on an industrial PL/SQL program. Procedures of this program are indirectly dependent on each other due to commonly accessed database elements. These dependencies are captured in the form of a hypergraph model. Clustering of this model revealed a packaging structure for the procedures. The accuracy of this structure was evaluated by domain experts. The accuracy was siginificantly higher with respect to the results obtained by clustering design structure matrices that are derived for the same subject system.

Acknowledgements. We thank the software developers and managers at Turkcell for sharing their code base with us and supporting our analysis.

References

1. Oracle Database Online Documentation 11g Release developing and using stored procedures. http://docs.oracle.com/, accessed in March 2016

[8] The approach is reevaluated for the new version of the subject system.

2. Altinisik, M., Sozer, H.: Automated procedure clustering for reverse engineering PL/SQL programs. In: Proceedings of the 31st ACM Symposium on Applied Computing. pp. 1440–1445 (2016)
3. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 3 edn. (2003)
4. Callo, T., America, P., Avgeriou, P.: A top-down approach to construct execution views of a large software-intensive system. Journal of Software: Evolution and Process 25(3), 233–260 (2013)
5. Çatalyürek, Ü.V., Kaya, K., Langguth, J., Uçar, B.: A partitioning-based divisive clustering technique for maximizing the modularity. In: Proceedings of the 10th DIMACS Implementation Challenge Workshop - Graph Partitioning and Graph Clustering. pp. 171–186 (2012)
6. Çatalyürek, U., Kaya, K., Langguth, J., Uçar, B.: A partitioning-based divisive clustering technique for maximizing the modularity. In: Bader, D.A., Meyerhenke, H., Sanders, P., Wagner, D. (eds.) Graph Partitioning and Graph Clustering. Contemporary Mathematics, AMS (2012)
7. Chaparro, O., Aponte, J., Ortega, F., Marcus, A.: Towards the automatic extraction of structural business rules from legacy databases. In: Proceedings of the 19th Working Conference on Reverse Engineering. pp. 479–488 (2012)
8. Ducasse, S., Pollet, D.: Software architecture reconstruction: A process-oriented taxonomy. IEEE Transactions on Software Engineering 35(4), 573 – 591 (2009)
9. Eppinger, S., Browning, T.: Design Structure Matrix Methods and Applications. MIT Press, Cambridge, MA, USA (2012)
10. Guo, G., Atlee, J., Kazman, R.: A software architecture reconstruction method. In: Proceedings of the 1st Working Conference on Software Architecture. pp. 15–34 (1999)
11. Habringer, M., Moser, M., Pichler, J.: Reverse engineering PL/SQL legacy code: An experience report. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution. pp. 553–556 (2014)
12. L. Qingshan et al.: Architecture recovery and abstraction from the perspective of processes. In: WCRE. pp. 57–66 (2005)
13. Mitchell, B., Mancoridis, S.: On the automatic modularization of software systems using the bunch tool. IEEE Trans. Software Engineering 32(3), 193 – 208 (2006)
14. Murphy, G., Notkin, D., Sullivan, K.: Software reflexion models: Bridging the gap between design and implementation. IEEE Transactions on Software Engineering 27(4), 364 – 308 (2001)
15. P. Clements et al.: Documenting Software Architectures. Addison-Wesley (2002)
16. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM 15(12), 1053–1058 (1972)
17. S. Eick et al.: Does code decay? assessing the evidence from change management data. IEEE Transactions on Software Engineering 27(1), 1 – 12 (2001)
18. Sangal, N., Jordan, E., Sinha, V., Jackson, D.: Using dependency models to manage complex software architecture. In: Proceedings of the 20th Conference on Object-Oriented Programming, Systems, Languages and Applications. pp. 167–176 (2005)
19. Sangwan, R., Neill, C.: Characterizing essential and incidental complexity in software architectures. In: Proceedings of the 3rd European Conference on Software Architecture. pp. 265–268 (2009)
20. Sullivan, K., Cai, Y., Hallen, B., Griswold, W.: The structure and value of modularity in software design. In: Proceedings of the 8th European Software Engineering Conference. pp. 99–108 (2001)
21. Sun, C., Zhou, J., Cao, J., Jin, M., Liu, C., Shen, Y.: ReArchJBs: a tool for automated software architecture recovery of javabeans-based applications. In: Proceedings of the 16th Australian Software Engineering Conference. pp. 270–280 (2005)
22. Taylor, R., Medvidovic, N., Dashofy, E.: Software Architecture: Foundations, Theory, and Practice (2009)