Static Analysis of Data-Intensive Applications

Csaba Nagy
University of Szeged
Department of Software Engineering
Dugonics tér 13. H-6725 Szeged, Hungary
ncsaba@inf.u-szeged.hu

***Abstract*—Data-intensive systems are designed to handle data at massive scale, and during the years they might evolve to very large, complex systems. In order to support maintenance tasks of these systems several techniques have been developed to analyze the source code of applications or to analyze the underlying databases for the purpose of reverse engineering, e.g. quality assurance or program comprehension. However, only a few of these techniques take into account the specialties of data-intensive systems.**

In this thesis we conducted research to analyze and to improve data-intensive applications via different methods based on static analysis: methods for recovering architecture of data-intensive systems and a quality assurance methodology for applications developed in Magic 4GL. We targeted SQL as the most widespread databases are relational databases using certain dialect of SQL for their queries. With the proposed techniques we were able to analyze large scale industrial projects, such as banking systems with more than 3 million lines of code, and we successfully recovered architecture maps and quality issues of these systems.

***Keywords*-Data-intensive systems, program dependencies, program analysis, CRUD matrix, SQL extraction, Magic 4GL**

I. Introduction

Data-intensive systems are are usually constructed of one or more databases and some applications communicating with these databases. These sort of systems are increasingly popular as databases play important role in many architectures. Today, these applications are part of our daily life (e.g. ERPs, CRMs). Such a system usually has a complex, sometimes even chaotic architecture with a large complex code base. Reverse engineering techniques have been widely used to support maintaining these systems in many different fields such as program comprehension, migration and testing.

In this thesis we utilize static analysis techniques with the goal to support program comprehension and improve software quality for data-intensive systems.

We conducted research in different fields which we introduce in this paper, hence the main contributions are:

- an SQL query extraction technique which can be utilized for further analyses of embedded SQL queries [1],
- a method to compute dependencies via data accesses in data-intensive systems [1],
- a method to use data dependencies to recover architecture of legacy database applications [2],
- a quality assurance methodology for applications developed in Magic 4GL [3], [4], [5], [6].

Most of the introduced analyses techniques are based on the Columbus reverse engineering technology developed at the Department of Software Engineering, University of Szeged in co-operation with FrontEndART Ltd. Columbus was first introduced to analyze C++ code for quality assurance purposes [7] and since then it has many different uses in the fields of software quality assessment, software quality improvement, software comprehension, and monitoring software development life-cycle attributes.

In previous works we extended the reverse engineering tool set with a front-end to analyze different dialects (Oracle, Transact) of SQL. This front-end serves as the basis of our further analyses.

II. Extracting embedded SQL from source code

Relational databases (RDBMS) are the most common database management systems used in data-intensive applications. The typical way of communicating with an RDBMS is to use SQL queries through a library such as JDBC. ORM technologies (e.g. Hibernate) are becoming popular too, but at lower level they also use SQL statements. Hence, most of the reverse engineering methods heavily depend on the extraction or capturing of SQL statements used to communicate with the underlying RDBMS.

Depending on the client side of the application, SQL statements can be embedded into the source code in a hard-coded way (e.g. in constant variables, string literals) or they can be constructed dynamically through string expressions, for instance.

Many approaches have been proposed to analyze embedded SQL statements via static and dynamic analysis techniques as well. Static analysis techniques are typically based on string analysis [8], [9] which require an in-depth data or control flow analysis. These techniques have their main advantage that they analyze the full source, but they might be unable to analyze dynamically constructed statements. Dynamic analysis techniques capture the statements being sent to the database at run-time, but they depend on the actual execution and input, so they might miss not executed cases. Cleve presents a summary on these techniques in his thesis [10]. Cordy *et al.* published papers [11], [12] describing the TXL language and its applications including embedded SQL parsing.

In our paper [1] we introduced an extraction technique to analyze SQL statements embedded in the source code of a special procedural language. The programming style of this

language makes the whole system strongly database dependent and it makes the use of SQL queries common in the system. The SQL statements to be executed are embedded as strings sent to specific library procedures and their results can be stored in given variables. This method is actually the same as that for procedural languages where embedded queries are sent to the database via libraries like JDBC. This makes our method general and suitable for other languages too.

The implemented approach is based on the simple idea of substituting the unrecognized query fragments in a string concatenation with special substrings. For instance, it is possible to simply replace the `name` variable with a string "@@name@@". If the SQL parser is able to handle this string as an identifier, then the received query string will be a syntactically correct SQL command (see Figure 1). With this simple idea we need to locate the library procedures sending SQL commands to the database in order to perform the string concatenation, and the above-mentioned substitution of variable, procedure name and other source elements. Whenever the constructed string is considered syntactically correct, it has the main characteristics of the executed SQL command.

```
SELECT firstname, lastname
  FROM @@customer_table@@
  WHERE firstname
    LIKE('%@@name@@%');
```

Figure 1. Sample code of an extracted SQL command where the table name is determined by a variable.

Developers usually like to prepare statements as close to their execution place as possible and they prefer to keep SQL keywords in separate string literals. In most cases, it is possible to substitute the variables with their last defined values within the same control block. In other cases the variable can be replaced with the variable name as we describe it before.

The technique has its limitations, however in the context of ForrasSQL it worked reasonably well. With this technique we identified $7,434$ embedded SQL strings (based on the specific SQL library procedure calls) in a 315 kLOC application and we successfully analyzed $6,499$ SQL statements, which is 87% of all the embedded SQL strings.

III. Dependencies via Data Accesses in Data-Intensive Systems

The analysis of SQL queries can be utilized to discover dependencies in the software which arise through the database. Such dependencies can help us in tracking the flow of data or discovering explicit or implicit relations between source elements. Primary uses of these techniques are change impact analysis or architecture reverse engineering.

It has been previously shown that $CRUD$ matrices are useful tools to support program comprehension and quality assessment [13], [14]. In our paper [1] we show the application of a $CRUD$-based Usage Matrix for dependency analysis between program elements (a sample graph representation of a $CRUD$-based relations can be seen in Figure 2). In a large

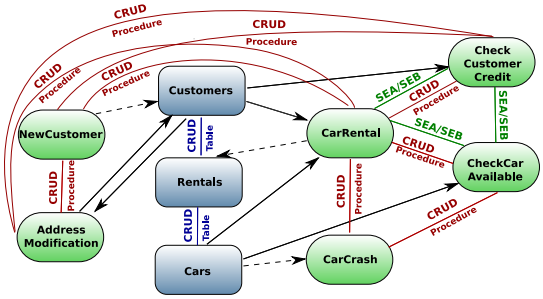

Figure 2. Typical $CRUD$ and SEA/SEB relations between procedures and between tables.

industrial system (the same that we described in the previous section) we identified relations between procedures based on table or column accesses and compared these relations to dependencies recovered by SEA/SEB relations [15]. The results showed that the disjoint parts of the relation sets of the two methods were similar in size, and that their intersection was considerably smaller (about 3% of the union). Based on this empirical evaluation, we concluded that neither of the relations was definitely better (safer and more precise) than the other; they were simply different. Thus they should be applied together in situations where a safe result is sought in the context of data-intensive systems.

Recently Liu et al. published a similar technique and they implement it for PHP-based database applications [16].

IV. Architecture Recovery of Legacy Database Applications

One potential use of previous techniques is to recover architecture of legacy data-intensive systems. In our previous work [2] one of our industrial partner asked us to help them in maintenance issues of their huge database system. They had a large Oracle PL/SQL system which evolved through the years to a system having a dump with more than 4.1 MLOC (data excluded, non-empty and non-comment lines of code). The system had more than 8,000 PL/SQL objects (tables, views, triggers, packages, routines). During the years, some implementation tasks were outsourced to small companies and the development team of the company found itself in a situation that they could not maintain the system alone.

Utilizing our previous techniques we reconstructed the top level architecture map of their system based on low-level static analysis. We identified high-level components and their related objects during interviews of the development team and we recovered the relations (based on call and CRUD dependencies) between these components. Final results showed that each of the 26 logical component had relations to almost all other components (see Figure 3).

Identified dependencies also supported the elimination of a huge component from the PL/SQL source base, which they re-implemented in Java. With the help of the analysis, they could cut relations between the unused component and others.

Besides architectural issues with static code checkers and a clone detector we identified a number of coding issues which

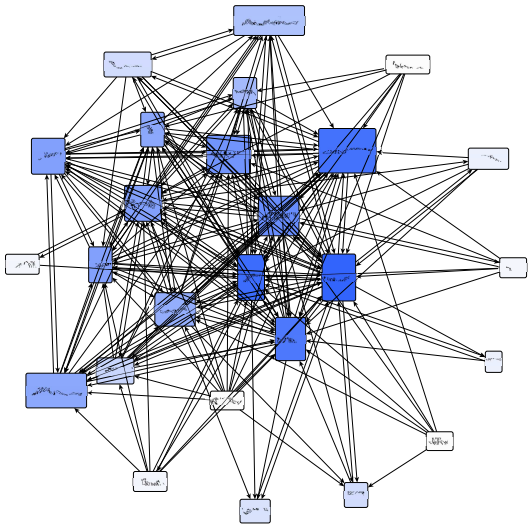

Figure 3. Relations between components (names distorted).

helped the company improving the quality of the code.

V. Quality Assurance of Applications Developed in Magic 4GL

Fourth generation languages (4GLs) are also referred to as Very High Level Languages (VLLs). A developer who develops an application in such a language does not need to write 'source code', but he can program his application at a higher level of abstraction and higher statement level, usually with the help of an application development environment.

Magic 4GL was introduced by Magic Software Enterprises (MSE) in the early 80's as an innovative technology to move from code generation to the use of an underlying meta model within an application generator. It was invented for the development of business applications with a special development style which is strongly based on a database. Most of the software elements are related to database entities: fields of records can be simply reached via variables and a task (which is the most similar element to a procedure) is always connected to a data table on which it performs operations. Hence, applications developed in Magic are also considered as data-intensive systems.

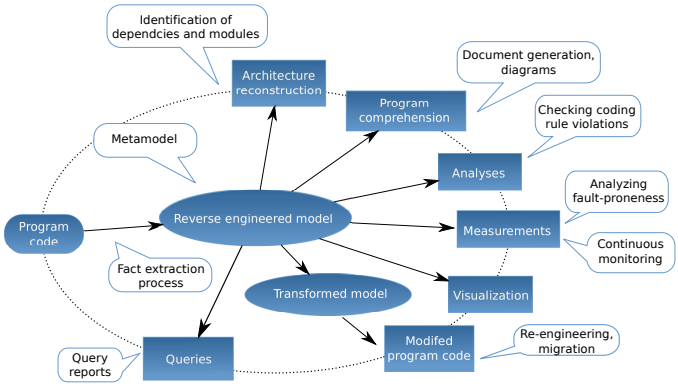

Figure 4. Columbus methodology adapted in the Magic environment.

With an industrial partner, SZEGED Software Inc., we adapted the Columbus technology for Magic [3]. The primary goal was to improve quality of Magic products via continuous monitoring of the development. We had to deal with many specialties of the language, but finally the outcome was a full-fledged system (***MAGISTER***) that provided useful services for the developers and also users of Magic products. ***MAGISTER*** is able to perform conventional quality assurance tasks (software code measurements using the product metric, coding conventions, violations control, etc.) and it has a user interface (SourceInventory) that allows developers and project managers to readily evaluate the data obtained from reports, diagrams and so on.

During the first development phase of ***MAGISTER*** we identified similarities to previous researches for data-intensive systems. As these systems had been developed since mid-1980s they evolved with the development of new features and sometimes by following the evolution of the underlying Magic Runtime Environment. However, MSE sometimes introduced major new features which was not affordable to implement in a large Magic application. (E.g. when Magic started to support Microsoft Windows.) With our industrial partner we conducted research in recovering the architecture of their logistical wholesale system [4]. The reconstructed architecture map was created from similar (call, CRUD, ...) relations as we used in the previous PL/SQL system, but with Magic components. The recovered architectural information was used to support a migration project from an older version of Magic (running on DOS) to the newest one (supporting modern, Rich Internet Application technologies).

To adapt to specialties of Magic we also had to implement a special complexity measure [5] and in recent research we extended ***MAGISTER*** with the support of automatic GUI testing [6]. In this research we exploit the specialty of Magic that it stores the layout of its user interfaces in its source code in a static (hard coded) way.

VI. Related Research

In previous sections we already cited related work, however reverse engineering data-intensive applications has many additional different applications, which must be mentioned here to have a full view on the static analysis of these systems. Here we present a brief overview on these topics.

a) Database reverse engineering: Reverse engineering databases can be considered as a different field, but it has some common points with data-intensive systems. Hainaut published a book on this topic [17] and Henrard wrote his PhD thesis on it [18]. Both sum up previous work in this area.

b) Program analysis and transformation: Cleve wrote his PhD thesis [19] on program analysis and transformation for data-intensive system evolution [10] mostly in the area of dependency analysis and migration to support data-intensive system evolution.

c) Testing database applications: There are a number of papers published in this area in different fields such as test input generation [20], [21], [22], test case generation [23], test coverage measurement [24] and regression testing [25].

d) Source Quality: Static analysis has been used before by Pantos *et al.* for source code-based quality assessment of ForrasSQL [26]. Brink *et al.* [14] used Usage Matrix to calculate metrics for applications with embedded SQL. Wassermann *et al.*, Gould *et al.* and Brass *et al.* published papers [8], [27], [28] in the area of static code checking of embedded SQL queries.

e) Impact analysis: Example topics of impact analysis in database-intensive systems are the analyzes of schema changes [29], [30] and supporting test case selection (e.g. in regression testing) [31].

VII. Conclusions

In this thesis we presented techniques to show how different fields of static analysis can be utilized by reverse engineering methods in the context of data-intensive systems. As results of these research, we show that the simple fact that such a system is designed in a database-centric way can support quality assessment, program comprehension, design or architecture recovery and testing among many other fields. With the presented techniques we successfully analyzed large-scale industrial projects written in ForrasSQL (~300 kLOC), Oracle PL/SQL (~3,000 kLOC) and Magic 4GL (~10,000 tasks). Thanks to the analysis of relations via databases we successfully created architecture maps, supported dead code elimination and identified coding issues.

References

[1] C. Nagy, J. Pantos, T. Gergely, and A. Beszedes, "Towards a safe method for computing dependencies in database-intensive systems," in *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2010, pp. 166–175.

[2] C. Nagy, R. Ferenc, and T. Bakota, "A True Story of Refactoring a Large Oracle PL/SQL Banking System," in *Industrial Track of the 8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2011)*, 2011.

[3] C. Nagy, L. Vidacs, R. Ferenc, T. Gyimothy, F. Kocsis, and I. Kovacs, "MAGISTER: Quality assurance of Magic applications for software developers and end users," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*. IEEE Computer Society, 2010, pp. 1–6.

[4] ——, "Solutions for reverse engineering 4gl applications, recovering the design of a logistical wholesale system," in *Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2011, pp. 343–346.

[5] C. Nagy, L. Vidács, R. Ferenc, T. Gyimóthy, F. Kocsis, and I. Kovács, "Complexity measures in 4gl environment," in *Proceedings of the 2011 international conference on Computational science and Its applications - Volume Part V*. Springer-Verlag, 2011, pp. 293–309.

[6] D. Fritsi, C. Nagy, R. Ferenc, and G. Tibor, "A layout independent gui test automation tool for applications developed in magic/unipaas," in *Proceedings of the 12th Symposium on Programming Languages and Software Tools*, 2011, pp. 248–259.

[7] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy, "Columbus – Reverse Engineering Tool and Schema for C++," in *Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002)*. IEEE Computer Society, Oct. 2002, pp. 172–181.

[8] G. Wassermann, C. Gould, Z. Su, and P. Devanbu, "Static checking of dynamically generated queries in database applications," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, Sep. 2007.

[9] A. S. Christensen, A. Müller, and M. I. Schwartzbach, "Precise analysis of string expressions," in *Proceedings of the 10th International Static Analysis Symposium, SAS'03*. Springer-Verlag, 2003, pp. 1–18.

[10] A. Cleve, "Program analysis and transformation for data-intensive system evolution," Ph.D. dissertation, University of Namur, 2003.

[11] J. R. Cordy, "The TXL source transformation language," *Science of Computer Programming*, vol. 61, no. 3, pp. 190–210, 2006.

[12] T. R. Dean, J. R. Cordy, A. J. Malton, and K. A. Schneider, "Agile parsing in TXL," *Automated Software Engineering*, vol. 10, no. 4, pp. 311–336, Oct. 2003.

[13] A. Van Deursen and T. Kuipers, "Rapid system understanding: Two COBOL case studies," in *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*. IEEE Computer Society, 1998, p. 90.

[14] H. v. d. Brink, R. v. d. Leek, and J. Visser, "Quality assessment for embedded SQL," in *SCAM '07: Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation*. IEEE Computer Society, 2007, pp. 163–170.

[15] J. Jász, Á. Beszédes, T. Gyimóthy, and V. Rajlich, "StaticExecute After/Before as a Replacement of Traditional Software Dependencies," in *Proceedings of the 2008 IEEE International Conference on Software Maintenance (ICSM'08)*. IEEE Computer Society, 2008, pp. 137–146.

[16] K. Liu, H. B. K. Tan, and X. Chen, "Extraction of attribute dependency graph from database applications," in *Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference*. IEEE Computer Society, 2011, pp. 138–145.

[17] J. luc Hainaut, "Introduction to database reverse engineering," Problems, Methods and Tools, Tutorial notes, CAiSE'95, Tech. Rep., 2002.

[18] J. Henrard, "Program understanding in database reverse engineering," Ph.D. dissertation, University of Namur, 2003.

[19] A. Cleve, T. Mens, and J.-L. Hainaut, "Data-intensive system evolution," *Computer*, vol. 43, no. 8, pp. 110–112, Aug. 2010.

[20] M. Marcozzi, W. Vanhoof, and J.-L. Hainaut, "Test input generation for database programs using relational constraints," in *Proceedings of the Fifth International Workshop on Testing Database Systems*. ACM, 2012, pp. 6:1–6:6.

[21] K. Pan, X. Wu, and T. Xie, "Generating program inputs for database application testing," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 73–82.

[22] M. Emmi, R. Majumdar, and K. Sen, "Dynamic test input generation for database applications," in *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 2007, pp. 151–162.

[23] M. Y. Chan and S. C. Cheung, "Testing database applications with sql semantics," in *Proceedings of the 2nd International Symposium on Cooperative Database Systems for Advanced Applications*. Springer, 1999, pp. 363–374.

[24] M. J. Suárez-Cabal and J. Tuya, "Using an sql coverage measurement for testing database applications," in *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*. ACM, 2004, pp. 253–262.

[25] F. Haftmann, D. Kossmann, and E. Lo, "A framework for efficient regression tests on database applications," *The VLDB Journal*, vol. 16, no. 1, pp. 145–164, Jan. 2007.

[26] J. Pantos, A. Beszedes, P. Gyenizse, and T. Gyimothy, "Experiences in adapting a source code-based quality assessment technology," in *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2008, pp. 311–313.

[27] C. Gould, Z. Su, and P. T. Devanbu, "Static checking of dynamically generated queries in database applications." in *Proceedings of the 26th International Conference on Software Engineering, ICSE 2004*, A. Finkelstein, J. Estublier, and D. S. Rosenblum, Eds. IEEE Computer Society, 2004, pp. 645–654.

[28] S. Brass and C. Goldberg, "Semantic errors in sql queries: A quite complete list," *J. Syst. Softw.*, vol. 79, no. 5, pp. 630–644, May 2006.

[29] S. K. Gardikiotis and N. Malevris, "A two-folded impact analysis of schema changes on database applications," *International Journal of Automation and Computing*, vol. 6, no. 2, pp. 109–123, 2009.

[30] A. Maule, W. Emmerich, and D. S. Rosenblum, "Impact analysis of database schema changes," in *ICSE '08: Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 451–460.

[31] A. Orso, T. Apiwattanapong, and M. J. Harrold, "Leveraging field data for impact analysis and regression testing," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 5, pp. 128–137, Sep. 2003.