A Static Concept Location Technique for Data-Intensive Systems: „Where Was This SQL Query Executed?"

Csaba Nagy, Anthony Cleve
PReCISE Research Center, University of Namur, Belgium
{csaba.nagy, anthony.cleve}@unamur.be

1 Introduction

An evolving software system is incrementally modified, changed by its developers during the development and maintenance phases [1]. Before the developers start working on a change they need to identify which parts of the source code implement the feature, and should be touched first during the change. In practice, what they do is a concept location task (also known as feature identification/location) which is „*the process that identifies where a software system implements a specific concept*" [2].

There are many existing approaches to support developers in concept location tasks starting from simple pattern matching (so-called 'grep' techniques) to more sophisticated methods like IR-based techniques or dependency analyzes [3]. However, none of the existing approaches consider when there is a database in the architecture, which adds further source artifacts or dependencies.

Here, we investigate a concept location approach for data-intensive systems, as applications with at least one database server in their architecture which is intensively used by its clients. Specifically, we introduce a static technique to identify the location(s) in the source code where a given SQL query was potentially sent to the database server.

2 Motivation

Identifying the location in the source code where a given SQL query was sent to the database is a regular debugging task of data-intensive systems. Typical scenarios are when queries need to be optimized for performance, or when they cause failures (e.g. a syntactic error or a deadlock issue). Complexity of the system or the use of ORM technologies can even complicate this tasks.

With dynamic analysis, it is possible to trace the query on the database side or on the client side too. At the database this is usually just a logging configuration, while on the client side they usually exploit that SQL queries are sent to the server via certain API calls which can be wrapped or hooked to catch the query.

However, dynamic analysis cannot help us in some situations. Suppose, that the user of the application experiences performance issues at the database; he identifies the query which causes the performance drop back in the log files of the database and sends us a bug report. Since the problem occurred at the database and was reported by it (client was not directly affected), we do not have a stack trace in the bug report. How can we spot out then, where the query was prepared in the source code? We must reproduce everything exactly as the user did which might be even impossible if we depend on the data stored in the database (perhaps we cannot even ask it for privacy reasons). In such situations, a static approach could provide us a great help in the concept location task.

3 Approach

Our approach, to identify the location in the source code where a given SQL query was sent to the database server, can be divided into three main steps (see Figure 1):

1. We extract the embedded SQLs from the source files with a technique which substitutes unrecognized code fragments with special identifiers [4]. The output of this step is a set of embedded SQLs which are prepared in the client code and potentially sent to the database.

2. We parse the extracted queries with a robust parser (which is able to handle the unrecognized code fragments). In the same step, we parse the database schema as well, and the queries that we are actually looking for. The output of the parser is an ASG (Abstract Syntax Graph) containing all the SQL statements.

3. We run a sort of tree-matching algorithm on the ASG to identify subtrees (queries) matching the trees of those statements that we are looking for. The output is a set of source code positions where the queries were potentially sent to the database server.

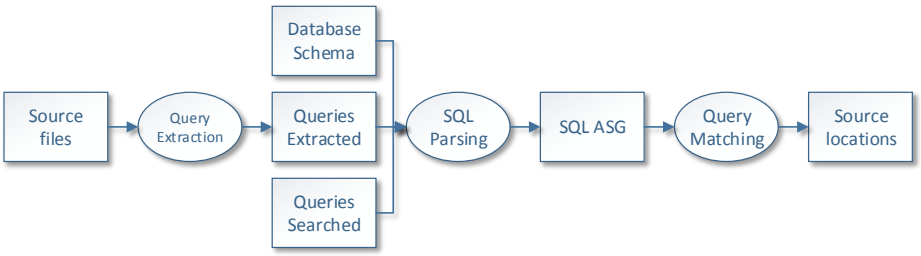

Figure 1: Overview of the approach

4 Current Results and Future Plans

We implemented our approach for systems written in Java and accessing the database through JDBC and/or Hibernate. To validate our approach, we test the implementation on the open source OSCAR EMR Clinical Management System. OSCAR accesses the database through JDBC and Hibernate and as a system with about 400 kLOC working with more than 400 tables, it is perfect to demonstrate the possibilities of our approach. Currently, we are in implementation/testing phases and able to extract JDBC queries, parse them supporting MySQL dialect and match the extracted queries to the concrete ones. We plan to extend our approach to handle Hibernate as well, where a key challenge is that a query can be specified as an HQL or Criteria query too, which is then compiled to the query language of the database server.

References

[1] V. Rajlich and P. Gosavi, "Incremental change in object-oriented programming," *IEEE Softw.*, vol. 21, no. 4, pp. 62–69, Jul. 2004.

[2] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev, "Static techniques for concept location in object-oriented code," in *Proc. of the 13th International Workshop on Program Comprehension (IWPC'05)*. IEEE Computer Society, 2005, pp. 33–42.

[3] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[4] L. Meurice, J. Bermudez, J. Weber, and A. Cleve, "Establishing referential integrity in legacy information systems - reality bites!" in *Proc. of 30th Interntional Conference on Software Maintenance and Evolution (ICSME)*. Victoria, BC, Canada: IEEE Computer Society, Oct. 2014.