Mining SQL Execution Traces for Data Manipulation Behavior Recovery

Marco Mori*, Nesrine Noughi, and Anthony Cleve

PReCISE Research Center, University of Namur
{marco.mori,anthony.cleve,nesrine.noughi}@unamur.be

Abstract. Modern data-intensive software systems manipulate an increasing amount of heterogeneous data in order to support users in various execution contexts. Maintaining and evolving activities of such systems rely on an accurate documentation of their behavior which is often missing or outdated. Unfortunately, standard program analysis techniques are not always suitable for extracting the behavior of data-intensive systems which rely on more and more dynamic data access mechanisms which mainly consist in run-time interactions with a database. This paper proposes a framework to extract behavioral models from data-intensive program executions. The framework makes use of dynamic analysis techniques to capture and analyze SQL execution traces. It applies clustering techniques to identify data manipulation functions from such traces. Process mining techniques are then used to synthesize behavioral models.

Keywords: data-manipulation behavior recovery, data-oriented process mining, data-manipulation functions

1 Introduction

Data-intensive systems typically consists of a set of applications performing frequent and continuous interactions with a database. Maintaining and evolving data-intensive systems can be performed only after the system has been sufficiently understood, in terms of structure and behavior. In particular, it is necessary to recover missing documentation (models) about the data manipulation behavior of the applications, by analyzing their interactions with the database. In modern systems, such interactions usually rely on dynamic SQL, where automatically generated SQL queries are sent to the database server.

The literature includes various static and dynamic program analysis techniques to extract behavioral models from traditional software systems. Existing *static* analysis techniques [19,18,22,7,20], analyzing program *source code*, typically fail in producing complete behavioral models in presence of dynamic SQL. They cannot capture the dynamic aspects of the program-database interactions, influenced by context-dependent factors, user inputs and results of

* beneficiary of an FSR Incoming Post-doctoral Fellowship of the *Académie universitaire 'Louvain'*, co-funded by the Marie Curie Actions of the European Commission

preceding data accesses. Existing *dynamic* analysis techniques [10], analyzing program *executions*, have been designed for other purposes than data manipulation behavior extraction. Several authors have considered the analysis of SQL execution traces in support to data reverse engineering, service identification or performance monitoring [8,9,12,11,23]. Such techniques look very promising for recovering an approximation of data-intensive application behavior.

In this paper, we propose a framework to recover the data manipulation behavior of programs, starting from SQL execution traces. Our approach uses clustering to group the SQL queries that implement the same high-level data manipulation function, i.e., that are syntactically equal but with different input or output values. We then adopt classical process mining techniques to recover data manipulation processes. Our approach operates at the level of a *feature*, i.e., a software functionality as it can be perceived by the user. A feature corresponds to a *process* enabling different instances, i.e., *traces*, each performing possibly different interactions with a database.

The reminder of this paper presents in Section 2 our approach along with a tool-supported validation. Finally, Section 3 discusses related work and Section 4 ends the paper showing possible future directions.

Motivating Example. We consider an e-commerce web store for selling products in a world-wide area. The system provides a set of features requiring frequent and continuous interactions with the database by means of executing SQL statements. For instance, the feature for retrieving products (*view_products*) accesses information about categories, manufacturers and detailed product information. Which data are accessed at runtime depends on dynamic aspects of the system. For example, given that a certain feature instance retrieves the categories of products before accessing product information we can derive that it corresponds to a category-driven search. If a certain instance accesses manufacturer information before product information we analogously derive that it corresponds to a manufacturer-driven search. By capturing and mining the database interactions of multiple feature instances, it is possible to recover the actual data manipulation behavior of the feature, e.g., a process model with a variability point among two search criteria.

2 Data Manipulation Behavior Recovery

Our framework supports the extraction of the data manipulation behavior of programs by exploiting several artifacts (see Fig. 1). We assume the existence of a *logical* and possibly of a *conceptual schema* with a mapping between them. The *conceptual schema* is a platform-independent specification of the application domain concepts, their attributes and relationships. The *logical schema* contains objects (tables, columns and foreign keys) implementing abstract concepts over which queries are defined. The conceptual schema and the mapping to the logical schema can be either available, or they can be obtained via database reverse engineering techniques [13]. Queries defined over the logical schema materialize the interactions occurring between multiple executions (traces) of a feature and the

underlying database. Once the source code related to a feature has been identified [14], different techniques can capture SQL execution traces. Those techniques, compared in [9], range from using the DBMS log to sophisticated source code transformation. Among others, the approaches presented in [1,17] recover the link between SQL executions and source code locations through automated program instrumentation, while [6] makes use of tracing aspects to capture SQL execution traces without source code alteration. Once a sequence of queries is captured, it is necessary to identify the different traces, each corresponding to a feature instance. This problem has been tackled in the literature of specification mining by analyzing value-based dependencies of methods calls [3].

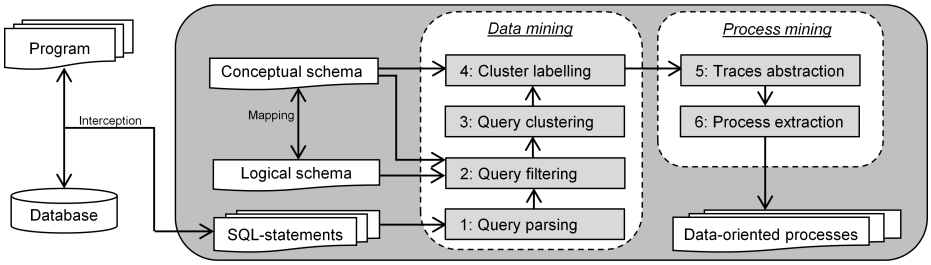


Fig. 1. Basics models: artifacts and components

Our approach is independent from the adopted trace capturing techniques. For each feature, it requires as minimal input a set of execution traces, each trace consisting of a sequence of SQL queries.

Query parsing (1). We characterize SQL queries according to (1) the information they recover or modify and (2) the related selection criteria. To this end, for each query we record a set of data-oriented properties according to the query type. For a *select* query we record a property with the *select* clause while for *delete*, *update*, *replace* or *insert* queries we record a property with the name of the table. If the query is either *update*, *replace* or *insert* we also record a property with the *set* clause and all its attributes. Finally for all query types but the *insert* we add a property for the *where* clauses along with their attributes. By means of these properties we ignore the actual values taken as input and produced as output by each query. Figure 2 shows three SQL traces along with their corresponding properties. For instance, query q_1 is a select query over attribute *Password* of *Customer* table (property p_1) and it contains a *where* clause with an equality condition over Id attribute (p_2); query q_3 is a *select* over attributes Id and $Price$ of $Product$ (property p_4), it contains two where clauses, i.e., a natural join between $Product.Id$ and $PCategory.Product\_Id$ (p_5) and an equality condition over $PCategory.Category\_Id$ attribute (p_6).

Query filtering (2). We remove from the input traces the queries that do not express end-user concepts, i.e., the ones referring to database system tables or log tables appearing only in the logical schema. In our example we remove q_{10}

Trace 1:

```
q1: SELECT Customer.Password FROM Customer WHERE Customer.Id = 'Mark27'; [p1,p2]
q2: SELECT Category.Id, Category.Image FROM Category; -> [p3]
q3: SELECT Product.Id, Product.Price FROM Product, PCategory WHERE Product.Id=PCategory.Product_Id AND
      PCategory.Category_Id='1'; -> [p4,p5,p6]
q4: SELECT PLang.Description FROM PLang, Language WHERE PLang.Language_Id=Language.Code AND PLang.Product_Id
      ='1A23' AND Language.Name='Italian'; -> [p7,p8,p9,p10]
q5: SELECT SpecialProduct.NewPrice FROM SpecialProduct,Product WHERE SpecialProduct.Product_Id=Product.Id
      AND Product.Id='1A23'; -> [p11,p12,p13]
q6: SELECT Manufacturer.Name FROM Manufacturer,Product WHERE Manufacturer.Id=Product.Manufacturer_Id AND
      Product.Id='1A23'; -> [p14,p15,p13]
q7: SELECT PLang.Description FROM PLang, Language WHERE PLang.Language_Id=Language.Code AND PLang.Product_Id
      ='1F32' AND Language.Name='Italian'; -> [p7,p8,p9,p10]
q8: SELECT SpecialProduct.NewPrice FROM SpecialProduct,Product WHERE SpecialProduct.Product_Id=Product.Id
      AND Product.Id='1F32'; -> [p11,p12,p13]
q9: SELECT Manufacturer.Name FROM Manufacturer,Product WHERE Manufacturer.Id=Product.Manufacturer_Id AND
      Product.Id='1F32'; -> [p14,p15,p13]
q10: INSERT INTO Log(IdEvent,Event,Date,Time) VALUES ('021','PrAcc1A23-1F32','2013-02-22','12:21:00'); -> [
      p16]
```

Trace 2:

```
q11: SELECT Customer.Password FROM Customer WHERE Customer.Id = 'JennyMa'; [p1,p2]
q12: SELECT Category.Id, Category.Image FROM Category; -> [p3]
q13: SELECT Product.Id, Product.Price FROM Product, PCategory WHERE Product.Id=PCategory.Product_Id AND
      PCategory.Category_Id='2'; -> [p4,p5,p6]
```

Trace 3:

```
q14: SELECT Customer.Password FROM Customer WHERE Customer.Id = 'DanWer'; [p1,p2]
q15: SELECT Manufacturer.Id, Manufacturer.Name FROM Manufacturer -> [p17]
q16: SELECT Product.Id, Product.Price FROM Product WHERE Product.Manufacturer_Id='AppleNamur01' -> [p4,p18]
q17: SELECT PLang.Description FROM PLang, Language WHERE PLang.Language_Id=Language.Code AND PLang.
      Product_Id='2D11' AND Language.Name='Italian'; -> [p7,p8,p9,p10]
q18: SELECT SpecialProduct.NewPrice FROM SpecialProduct,Product WHERE SpecialProduct.Product_Id=Product.Id
      AND Product.Id='2D11'; -> [p11,p12,p13]
q19: SELECT Manufacturer.Name FROM Manufacturer,Product WHERE Manufacturer.Id=Product.Manufacturer_Id AND
      Product.Id='2D11'; -> [p14,p15,p13]
q20: INSERT INTO Log(IdEvent,Event,Date,Time) VALUES ('022','PrAcc2D11','2013-02-28','14:00:03'); -> [p16]
```

SQL-statements properties:

```
p1="SELECT Customer.Password", p2="Customer.Id.EQ_VALUE", p3="SELECT Category.Id Category.Image",
p4="SELECT Product.Id Product.Price", p5="Product.Id=PCategory.Product_Id",
p6="PCategory.Category_Id.EQ_VALUE", p7="SELECT PLang.Description", p8="PLang.Language_Id=Language.Code",
p9="PLang.Product_Id.EQ_VALUE", p10="Language.Name.EQ_VALUE", p11="SELECT SpecialProduct.NewPrice",
p12="SpecialProduct.Product_Id=Product.Id", p13="Product.Id.EQ_VALUE", p14="SELECT Manufacturer.Name",
p15="Product.Manufacturer_Id=Manufacturer.Id",  p16="INSERT INTO Log",
p17=" SELECT Manufacturer.Id Manufacturer.Name", p18="Product.Manufacturer_Id.EQ_VALUE"
```

Fig. 2. Web Store: Traces of SQL statements with data-oriented properties

and q_{20} accessing table *Log* without a counterpart in the conceptual schema.
Query clustering (3). We cluster queries having the same data-oriented properties thus producing disjoint partitions, related to different database accesses. We report in Table 1 the clusters obtained from queries in Fig.2.

Table 1. Web Store: Clusters of SQL queries

C1	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**
$\{q_1, q_{11}, q_{14}\}$	$\{q_2, q_{12}\}$	$\{q_3, q_{13}\}$	$\{q_4, q_7, q_{17}\}$	$\{q_5, q_8, q_{18}\}$	$\{q_6, q_9, q_{19}\}$	$\{q_{15}\}$	$\{q_{16}\}$
$\{p_1, p_2\}$	$\{p_3\}$	$\{p_4, p_5, p_6\}$	$\{p_7, p_8, p_9, p_{10}\}$	$\{p_{11}, p_{12}, p_{13}\}$	$\{p_{13}, p_{14}, p_{15}\}$	$\{p_{17}\}$	$\{p_4, p_{18}\}$

Cluster labeling (4). We identify the data manipulation function implemented by each cluster by analyzing the conceptual schema fragment corresponding to the logical subschema accessed by the cluster queries. For determining the labels we adopt the same naming convection proposed in [5] to associate conceptual level operations to SQL query code. In addition, we associate the label with a set of input/output (I/O) parameters (see Table 2). Input parameters are

the attributes involved in equality or inequality conditions that appear in the data-oriented properties of the queries, while output parameters are the set of attributes appearing within the *select* query property.

Table 2. Web Store: Clusters with data manipulation functions and I/O parameters

Cluster	**Input**	**Output**
C1:*getCustomerById*	$\{Id\}$	$\{Password\}$
C2*getAllCategory*	–	$\{Id, Image\}$
C3:*getAllProductOfCategoryViaPCategory*	$\{Category\_Id\}$	$\{Id, Price\}$
C4:*getAllLanguageOfProductViaPLang*	$\{Product\_Id, Name\}$	$\{Description\}$
C5:*getSpecialProductOfProductViaSProd*	$\{Product.Id\}$	$\{NewPrice\}$
C6:*getManufacturerOfProductViaPManufact*	$\{Product.Id\}$	$\{Name\}$
C7:*getAllManufacturer*	–	$\{Id, Name\}$
C8:*getAllProductOfManufacturerViaPManufact*	$\{Manufacturer\_Id\}$	$\{Id, Price\}$

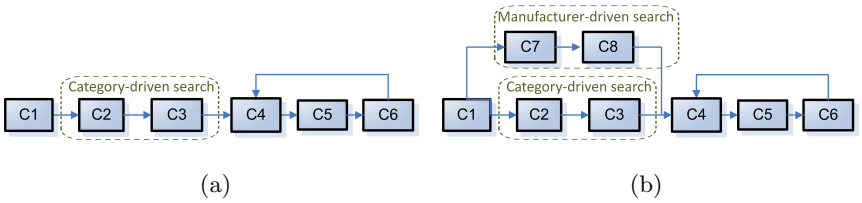
Process mining (5-6). We generate a process starting from a set of SQL traces of a single feature. The *traces abstraction* phase replaces SQL traces with the corresponding traces of data manipulation functions. The *process extraction* phase exploits a process mining algorithm to extract the feature behavior as a sequence of function executions with sequential, parallel and choice operators. In the following we show how to recover the data manipulation behavior of the *view_products* web-store feature starting from the traces of data manipulation functions in Table 3 (corresponding to queries in Fig.2).

Table 3. Web Store: Traces of data manipulation functions

Trace 1	*getCustomerById*(**C1**) - *getAllCategory*(**C2**) - *getAllProductOfCategoryViaPCategory*(**C3**) - *getAllLanguageOfProductViaPLang*(**C4**) - *getSpecialProductOfProductViaSProd*(**C5**) - *getManufacturerOfProductViaPManufact*(**C6**) - *getAllLanguageOfProductViaPLang*(**C4**) - *getSpecialProductOfProductViaSProd*(**C5**) - *getManufacturerOfProductViaPManufact*(**C6**)
Trace 2	*getCustomerById*(**C1**) - *getAllCategory*(**C2**) - *getAllProductOfCategoryViaPCategory*(**C3**)
Trace 3	*getCustomerById*(**C1**) - *getAllManufacturer*(**C7**) - *getAllProductOfManufacturerViaPManufact*(**C8**) - *getAllLanguageOfProductViaPLang*(**C4**) - *getSpecialProductOfProductViaSProd*(**C5**) - *getManufacturerOfProductViaPManufact*(**C6**)

Trace 1 gets customer information (C1), it performs a category-driven search of products by means of getting all the product categories (C2) and all the products of a certain selected category (C3). For each retrieved product, three functions are iterated: C4 retrieves product description, C5 extracts special product information and C6 extracts related manufacturer information. *Trace 2* is different from *Trace 1* because after function C3 no products are retrieved and the process ends. If we apply a mining algorithm to *Trace 1* and *2* we obtain a process (Fig. 3(a)) which performs consecutively functions C1, C2 and C3 before entering in the loop iterating C4, C5, and C6. The process ends after zero, one or more iterations of the loop. Let us now assume to include into the process *Trace 3* which is equal to *Trace 1* except that it searches products based on their manufacturer (functions C7 and C8) instead of searching by category (C2 and C3).

If we mine the process model by considering as input all the traces (Fig. 3(b)), we end up with a new alternative branch: the customer can now perform either a manufacturer-driven search or a category-driven search.



(a) (b)

Fig. 3. Web Store: process mined with (a) *Trace 1* and *2* and (b) *Trace 1*, *2* and *3*.

Tool support. The presented approach is implemented into an integrated tool which takes as input a set of SQL traces (each representing an instance of the same feature), the logical schema and optionally the conceptual schema and the conceptual-to-logical schema mapping. A SQL *parser* extracts the data-oriented properties while a *clustering* component exploits the *colibri-Java* Formal Concept Analysis tool[1] to cluster queries according to those properties. A *labeling* component generates data manipulation functions (i.e., cluster signatures) while a *trace abstraction* component uses a Java library[2] to create standardized event logs. Finally we rely on the de-facto standard process mining tool (*ProM* tool[3]) to create a Petri net from standardized event logs. *ProM* supports different process mining algorithms providing different trade-offs between completeness and noise [4] to be chosen according to specific application needs.

We applied our tool together with *ProM* and the ILP miner algorithm [21] (complete models with low noise) to extract data-oriented processes of a e-restaurant web application and we conducted a set of preliminary experiments to assess the sensitivity of our technique in producing correct processes depending on the traces log coverage. The tool supported the identification of correct features processes in a semi-automatic manner along with the help of the designer. A complete list of SQL statements grouped by feature with different traces, extracted data manipulation functions and corresponding processes are publicly available at the companion website[4]. The conceptual and logical schemas, accessible through the DB-MAIN[5] CASE tool, are also provided.

3 Related Work

In the literature different approaches use dynamic analysis of SQL queries with a different goal than data manipulation behavior understanding. The approaches

[1] `http://code.google.com/p/colibri-java/`
[2] `http://www.xes-standard.org/openxes/start`
[3] `http://www.promtools.org/`
[4] `http://info.fundp.ac.be/~mmo/MiningSQLTraces`
[5] DB-MAIN official website, `http://www.db-main.be`

presented in [8,9] analyze SQL statements in support to database reverse engineerinf, e.g., detecting implicit schema constructs [9] and implicit foreign keys [8]. The approach presented by Di Penta et al. [12] identifies services from SQL traces. The authors apply FCA techniques to name services I/O parameters thus supporting the migration towards Service Oriented Architecture. Debusmann et al. [11] present a dynamic analysis method for system performance monitoring, i.e., measuring the response time of queries sent to a remote database server. Yang et al. [23] support the recovery of a feature model by means of analyzing SQL traces. Although the former approaches analyze (particular aspects of) the data access behavior of running programs, none of the former approaches [8,9,12,11,23] is able to produce process models expressing such a behavior at a high abstraction level, as we do in this paper.

Other approaches (e.g., [16,15]) extract business processes by exploiting/combining static and dynamic analysis techniques, but they are not designed to deal with dynamically generated SQL queries. The most related approach, by Alalfi et al. [2], extracts scenario diagrams and UML security models by considering runtime database interactions and the state of the PHP program. These models are used for verifying security properties but they do not describe the generic data manipulation behavior of the program, they only analyze web-interface interactions. In addition they have not considered different possible instances of a given scenario as we claim it is necessary to extract a complete and meaningful model. Understanding processes starting from a set of execution traces is at the core of process mining. This paper does not make any additional contributions as far as process mining is concerned, but it is the first to apply such techniques to analyze program-database interactions.

4 Conclusion and future work

Our paper presented a tool-supported approach to recover the data manipulation behavior of data-intensive systems. The approach makes use of clustering, conceptualization and process mining techniques starting from SQL execution traces captured at runtime. The approach is independent from the type of systems considered, provided that a query interception phase is possible. It could, for instance, be applied to legacy cobol systems, Java systems with or without Object-Relational-Mapping technologies, or web applications written in PHP. As for future work we plan to enrich the input traces with multiple sources of information like user input, source code and queries results with the aim of identifying the conditions that characterize decision points within process models.

References

1. M. Alalfi, J. Cordy, , and T. Dean. WAFA: Fine-grained dynamic analysis of web applications. In *WSE 2009*, pages 41–50, 2009.
2. M. H. Alalfi, J. R. Cordy, and T. R. Dean. Recovering role-based access control security models from dynamic web applications. In *ICWE*, pages 121–136. 2012.

3. G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *ACM Sigplan Notices*, volume 37, pages 4–16, 2002.
4. J. Buijs, B. Dongen, and W. Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM*, volume 7565 of *LNCS*, pages 305–322, 2012.
5. A. Cleve, A.-F. Brogneaux, and J.-L. Hainaut. A conceptual approach to database applications evolution. In *ER*, pages 132–145, 2010.
6. A. Cleve and J.-L. Hainaut. Dynamic analysis of SQL statements for data-intensive applications reverse engineering. In *WCRE 2008*, pages 192–196, 2008.
7. A. Cleve, J. Henrard, and J.-L. Hainaut. Data reverse engineering using system dependency graphs. In *WCRE 2006*, pages 157–166, 2006.
8. A. Cleve, J.-R. Meurisse, and J.-L. Hainaut. Database semantics recovery through analysis of dynamic sql statements. *J. Data Semantics*, 15:130–157, 2011.
9. A. Cleve, N. Noughi, and J.-L. Hainaut. Dynamic program analysis for database reverse engineering. In *GTTSE*, pages 297–321, 2011.
10. B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Trans. Software Eng.*, 35(5):684–702, 2009.
11. M. Debusmann and K. Geihs. Efficient and transparent instrumentation of application components using an aspect-oriented approach. In *DSOM 2003*, volume 2867 of *LNCS*, pages 227–240, 2003.
12. C. D. Grosso, M. D. Penta, and I. G. R. de Guzmán. An approach for mining services in database oriented applications. In *CSMR*, pages 287–296, 2007.
13. J.-L. Hainaut, J. Henrard, V. Englebert, D. Roland, and J.-M. Hick. Database reverse engineering. In *Encyclopedia of Database Systems*, pages 723–728. Springer US, 2009.
14. H. Kazato, S. Hayashi, T. Kobayashi, T. Oshima, S. Okada, S. Miyata, T. Hoshino, and M. Saeki. Incremental feature location and identification in source code. In *CSMR*, pages 371–374, 2013.
15. Y. Labiche, B. Kolbah, and H. Mehrfard. Combining static and dynamic analyses to reverse-engineer scenario diagrams. In *ICSM*, pages 130–139, 2013.
16. H. R. M. Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event correlation for process discovery from web service interaction logs. *VLDB*, 20(3):417–444, 2011.
17. M. N. Ngo and H. B. K. Tan. Applying static analysis for automated extraction of database interactions in web applications. *Information and software technology*, 50(3):160–175, 2008.
18. J.-M. Petit, J. Kouloumdjian, J.-F. Boulicaut, and F. Toumani. Using queries to improve database reverse engineering. In *ER*, pages 369–386, 1994.
19. J. C. Silva, J. C. Campos, and J. Saraiva. Gui inspection from source code analysis. *ECEASST*, 33, 2010.
20. H. van den Brink, R. van der Leek, and J. Visser. Quality assessment for embedded sql. In *SCAM*, pages 163–170, 2007.
21. J. M. E. van derWerf, B. F. van Dongen, C. A. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. *Fundamenta Informaticae*, 94(3):387–412, 2009.
22. D. Willmor, S. M. Embury, and J. Shao. Program slicing in the presence of a database state. In *ICSM 2004*, pages 448–452, 2004.
23. Y. Yang, X. Peng, and W. Zhao. Domain feature model recovery from multiple applications using data access semantics and formal concept analysis. In *WCRE*, pages 215–224, 2009.