INSTITUT FÜR INFORMATIK

Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach

Christian Zirkelbach
and Wilhelm Hasselbring

Bericht Nr. 1901
Februar 2019
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Live Visualization of Database Behavior for Large Software Landscapes: The RACCOON Approach

Christian Zirkelbach
and Wilhelm Hasselbring

Bericht Nr. 1901
Februar 2019
ISSN 2192-6247

e-mail: {czi, akr, wha}@informatik.uni-kiel.de

Technical Report

Abstract

Databases are essential components within large software landscapes, since they are employed in almost every information system. Based on the growing complexity of software systems and a steadily increasing amount of data which is collected, processed, and stored in databases, it is difficult to obtain a live overview of these software landscapes. This often leads to an insufficient knowledge of the actual internal structure and behavior of employed databases. Furthermore, databases are often involved in performance issues within information systems.

A solution to these problems is employing live visualizations of databases and related communication from applications within the software landscape. These visualizations allow operators to understand their databases in detail and to analyze database queries performed by applications. Based on established visualization concepts like the *entity relationship diagrams* and the *3D city metaphor*, operators can be supported in the task of database comprehension. Established monitoring techniques, like dynamic and static analysis, can be used to capture necessary information from applications and databases.

In this paper, we present our live visualization approach of databases and associated communication for large software landscapes. Our visualization offers two different views – a landscape-level and a database-level perspective. The landscape-level perspective provides an overview of monitored applications and related databases. The database-level perspective reveals database schemas within a database, shows contained tables and relationships, and allows for the inspection of executed queries based on the monitoring information collected at runtime.

1 Introduction

Large software landscapes often consist of a large number of systems, applications, and communication links. Usually these systems employ databases, which provide well-defined interfaces for retrieving, storing, and processing application data. Databases are pivotal components in large software landscapes and are affected by the growing complexity and evolutionary progression of software systems. In combination with the steadily increasing amount of data, it is difficult to maintain a live overview of these software landscapes, database-related communication, and connected databases. This often leads to insufficient knowledge of the actual structure and behavior of related databases. Additionally, databases are often involved in performance issues within software systems [1]. Therefore, it is necessary to (i) monitor database queries invoked by applications, (ii) analyze connected databases, and (iii) present the gathered information in an appropriate visualization.

Software landscapes are often visualized with UML-based deployment diagrams [2] or modified versions to keep an overview of the landscape and included application systems [3]. Databases by contrast are commonly represented by entity relationship diagrams [4]. To the best of our knowledge, there exists no visualization approach, which offers a software landscape facilitating the communication between applications and databases on the one hand, and the internal structure and actual usage of databases on the other hand.

In [5] we already described problems caused by missing or inappropriate database monitoring and visualization and presented a first sketch of a solution approach. In this paper, we propose an improved live visualization approach, which employs a combination of two different, complementary representations, to support the comprehension process of databases and related communication for large software landscapes. Our visualization offers two different views – a landscape-level and a database-level perspective. This combination of visualizations enables operators to understand their databases in detail and analyze database queries invoked by applications at runtime. Based on accepted visualization concepts like the *entity relationship diagrams* and the *3D city metaphor* [6], operators can be supported in the task of database comprehension. We apply established analysis techniques, namely dynamic and static analysis, to capture necessary information from applications and databases. Possible scenarios of our approach are exploration and enhanced software landscape and database comprehension.

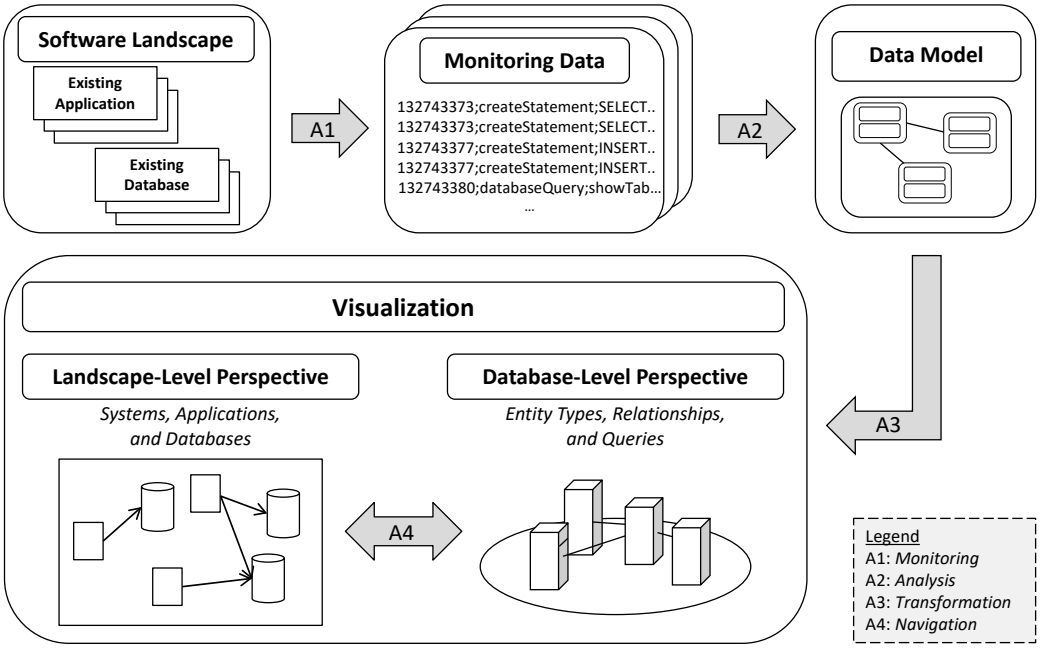


Figure 1: Overview of our RACCOON database behavior live visualization approach

In summary, our main contributions are:

- An interactive live monitoring and visualization approach for databases and related communication
- A UML- and data-flow-oriented landscape-level visualization providing an overview of applications and related databases
- A database-level visualization based on the *entity relationship diagrams* and the *3D city metaphor* showing the internal structure of a database and executed queries at runtime

The remainder of this paper is organized as follows. In Section 2, we describe our approach. Afterwards, we present an overview of our visualization in Section 3. In Section 4, we discuss related work. Finally, the conclusions are drawn and open questions are delineated in Section 5.

2 The Raccoon Approach

Our database behavior live visualization approach includes four consecutive activities (A1 to A4), which are briefly described in the following. Figure 1 illustrates an overview of the activities in our approach.

A1 – Monitoring: Within a software landscape existing applications and related databases are *monitored*. We collect conducted database queries in each application, detect which databases are used, and query these databases for structural information directly via their APIs. The results will be provided in form of a data stream, which contains monitoring logs (applications) and structural information (databases).

A2 – Analysis: In order to further process the monitoring information, we need to *analyze* it. Analyzing database queries contains basically two steps – (i) reconstructing the collected monitoring data into corresponding database queries and (ii) aggregating similar ones. For structural information gathered directly from databases, we take the results from the queried APIs and preprocess them for the next activity. Furthermore, we need to store the processed information. The result of this activity is a persistent data model for the whole software landscape, which contains the monitored applications, databases, and their communication among each other. This enables a live visualization of our reconstructed landscape and databases.

A3 – Transformation: To enable a visualization of our database queries and related databases, we need to *transform* the data model into a visualization model. This is a necessary step, as the visualization model should only contain relevant information for the requested perspective.

A4 – Navigation: Our live database visualization offers two perspectives. The operator is able to view and navigate between them. Each perspective is different and focuses on distinct use-cases – either getting an overview of the software landscape or detailed information of a specific database. Therefore, each perspective employs a different visualization metaphor. Thus, we provide separated views on the landscape-level and database-level within the software landscape. In the following section, we describe the two perspectives, their interaction capabilities, and the navigation between them in detail.

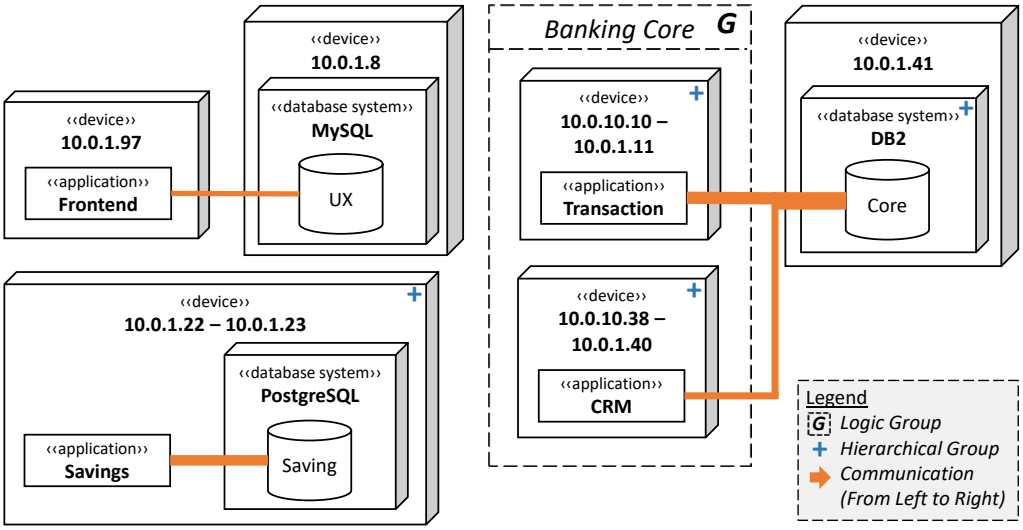


Figure 2: Landscape-level visualization: communication between systems, applications, and databases in a software landscape

3 Live Visualization of Database Behavior

In this section, we present our database behavior live visualization approach for large software landscapes. It consists of two, complementary perspectives, namely a landscape-level and a database-level perspective. The first perspective employs a UML-based 2D visualization of the reconstructed software landscape. The landscape includes monitored applications and databases and shows performed their interactions among each other. The second perspective offers a detailed view of a single database by showing its included tables, columns, and relationships. Additionally, executed database queries are visualized in form of connections between tables. Both perspectives avoid problematic color combinations and employ a color vision impaired color design as described in [7]. Thus, we only need one color scheme and support impaired operators from the beginning.

Operators need an appropriate visualization of databases for large software landscapes for comprehension and planning tasks. As databases are affected by evolutionary changes, it is necessary to keep track of changes in the database or queries invoked by applications. This situation requires a live monitoring of relevant applications and databases in order to update the visualizations based on monitoring information. Thus, the analysis rests upon two techniques – (i) dynamic analysis, which implies database communication invoked by applications during runtime, and (ii) static analysis, which

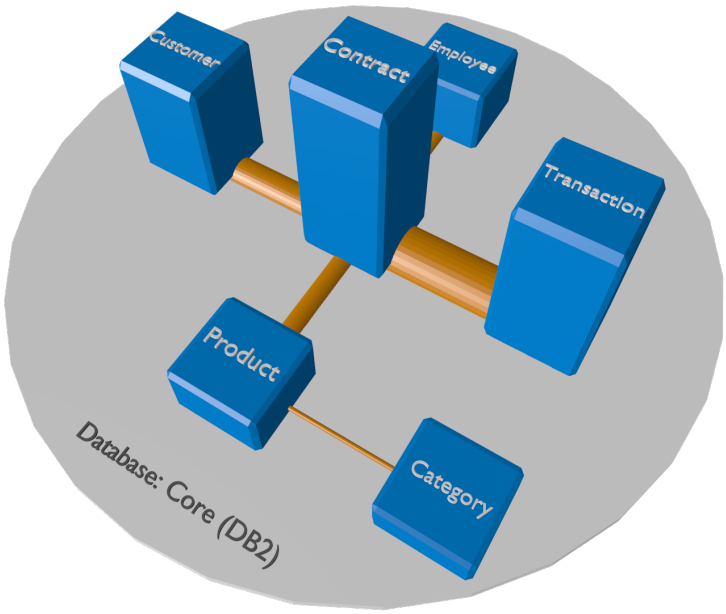
embodies the internal structure of a database, namely databases schemas and comprised objects like tables, columns, constraints, and statistical information.

In order to ease the comprehension process for operators, we provide a consistent layout as described in [8]. Changes detected by the monitoring are processed and stored in the data model, transformed into the visualization model, and finally illustrated in an updated visualization. In the following, we use the SQL (Structured Query Language) terminology.

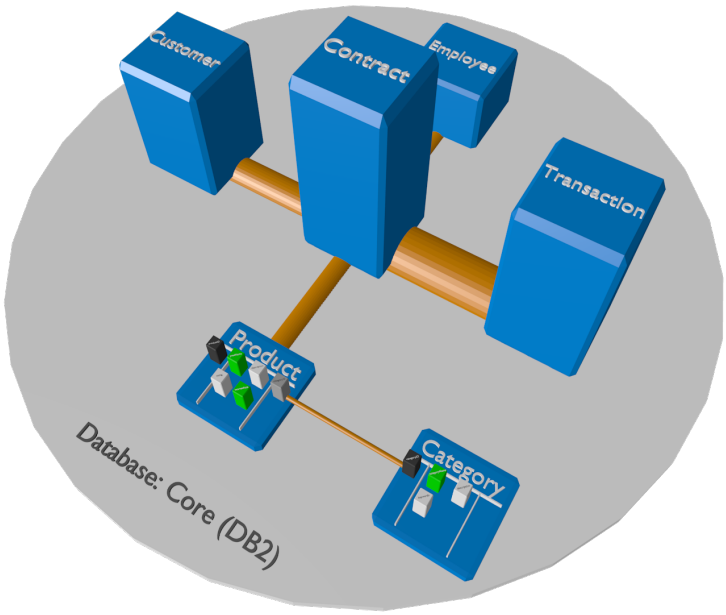
3.1 Landscape-Level Perspective

The landscape-level perspective is based upon the UML deployment diagram [2] and provides an overview of the reconstructed software landscape containing monitored applications and databases. Additionally, we employ a data-flow diagram direction-oriented layout [9], i.e., the data flows from the left (source) to the right (target). In order to distinguish between systems, applications, and databases, we employ UML stereotypes (‹‹stereotype››). A sketch of this visualization is displayed in Figure 2. It shows systems (‹‹device››), included monitored applications (‹‹application››), and database systems (‹‹database system» – showing the vendor) with contained databases (cylinder with the database name). Furthermore, we visualize the communication (edge – colored ■) between applications and databases. The thickness of an edge represents the amount of communication, i.e., the more database queries are executed, the thicker the edge. Once a new monitored object is observed, a respective visualization object is integrated into the landscape.

Additionally, we support an automatic visual clustering of similar objects like systems, applications, and databases into hierarchical groups (indicated by the symbol +). This allows us to provide a visual abstraction of monitored objects, which occur more than once within the landscape. By clicking on the symbol (+) the operator is able to reveal individuals of the hierarchical group. For example, the *‹‹database system›› DB2* is running multiple times within the *‹‹device›› 10.0.1.41* containing the database *Core*. Moreover, the operator is able to cluster devices, which contain applications and databases (represented by an dashed rectangle with a name and a ***G*** in the upper right corner), into logical groups. This feature allows him to add further semantics to the visualization for planning or documentation purposes, e.g., redeployments or migrations. Finally, the detailed view of a specific database (database-level perspective) is provided by clicking on the respective database within the landscape-level visualization.



(a) Database tables, relationships, and executed queries



(b) Details of the tables *Product* and *Category*

Figure 3: Mockup of the database-level visualization: tables, relationships, and executed queries in a single database (*Core (DB2)*)

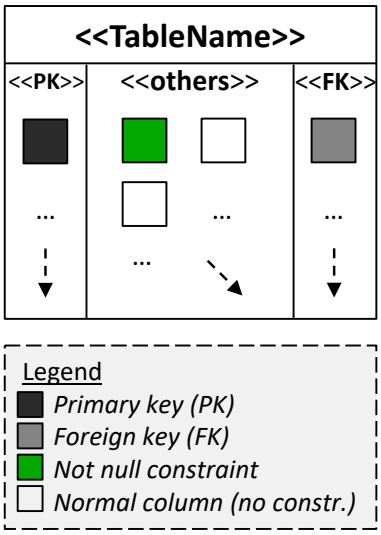


Figure 4: Semantics of an opened table: columns and constraints

3.2 Database-Level Perspective

The database-level perspective is based on the *3D city metaphor* [6], a well established representation in the area of software engineering, and the *entity relationship diagrams* [4], which is a commonly used visualization for relational databases. The visualization shows the internal structure and executed queries within a single database. Figure 3a sketches the database tables, relationships, and performed queries during runtime in the database *Core (DB 2)*. The *3D city metaphor* basically provides three different kinds of visualized objects (districts, buildings, and streets). In the following, we describe (i) what information is represented by these objects and (ii) how they behave.

Districts: Each database (schema) contains one root district. This district is displayed as a round layer with a fixed height (colored ■), as shown in Figure 3a, and acts as a foundation for included tables and relationships.

Buildings: Tables within our database (schema) form buildings within the *3D city metaphor* (e.g., the table *Product*). Each table is visualized as a rectangular box (colored ■) with a dynamic height. The default metric used for the height is based on the absolute number of rows stored in the table. It is also possible to apply other metrics, e.g., the number of times the table was involved in queries. Additionally, tables can be opened in order to reveal their columns, as shown in Figure 3b, and thus become districts. Fur-

thermore, columns within a table are visualized by small rectangular boxes (colored in several colors) with a fixed height. Figure 4 illustrates the semantics of the representation. We separate the columns of a table into three groups based on their constraints. The first group includes columns with a primary key (PK – colored ■). The second group contains columns with a foreign key (FK – colored ■). The third group comprises columns that either have a different constraint (e.g., not null – colored ■) or no constraint (colored □). The layout is based on a single-column (PK), multiple-column (others), and single-column (FK) alignment. Further columns are added in direction of the dashed arrow (PK and FK: south, others: south-east). In order to ease distinguishing between the groups, we employ dividers. Figure 5 shows a zoomed-in visualization of the opened table *Product*. It illustrates the contained and colored columns, separated by the dividers into the three groups mentioned above.

Streets: Streets represent the communication between tables, i.e., relationships and executed database queries during runtime. The visualization follows the representation of communication within the landscape-level perspective. Thus, they are shown as edges (colored ■) between tables and the thickness correlates with the current amount of executed queries. Once a table is opened, the edge connects the relevant columns (e.g., foreign keys in a table and their corresponding counterpart as a primary key in another table) instead of the tables (shown in Figure 3b). Hence, the operator can directly perceive the relation between two tables.

In order to get details for database queries (between tables), the operator can click on a communication edge. As a result, all queries, which contain the connected tables, are shown in a sortable list with detailed information like the number of executions, the response time, and the concrete executed statement. This feature is particularly interesting, when conducting database optimizations, finding performance issues [10], or evaluating database performance [11]. Another feature includes highlighting queries and tables. When the operator selects a specific query or edge, the involved tables are highlighted. Additionally, selecting a table or related edges (database queries) highlights involved objects for further analysis or comprehension. Finally, the operator is able to head back to the landscape-level perspective at any point within the visualization. In this way, he can review the software landscape or take a look at another database and vice versa.

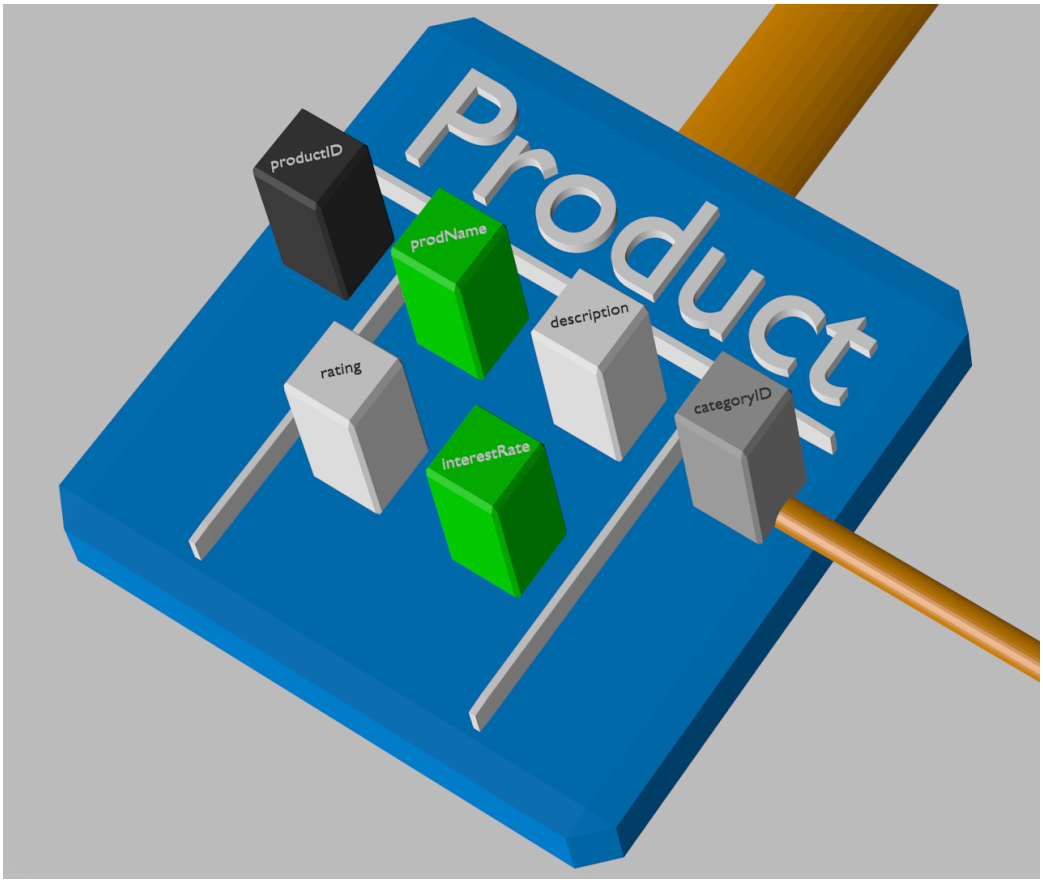


Figure 5: Visualized Opened table *Product* – showing columns, constraints, and relationships

4 Related Work

There are several approaches, which are related toward our envisioned approach, visualization, or research topic. Due to space restrictions, we only list closely related work focusing on the visualization. ExplorViz [3], [12], [13] is a web-based tool, which enables the monitoring and visualization of large software landscapes. Basically, it offers two different visualization options. The first visualization shows a reconstructed software landscape based on monitoring information. The second visualization offers an in-detail representation of a single application and involved packages, classes, and communication. In contrast, our approach focuses on the monitoring and visualization of databases and applications, which communicate with them, in order to facilitate software landscape and database comprehension. SynchroVis [14] visualizes monitoring traces in 3D with the city metaphor for analyzing concurrency. DAHLIA [15] is an interactive visualization tool, which facilitates to analyze the database usage in order to support software and database schema evolution. The tool collects snapshots of database schemas from a software repository based on static analysis and utilizes a 3D visualization for exploring the monitored evolution. Recently they released version 2.0, which includes the support for Object-Relational-Mapping frameworks [16]. Based on this feature, the tool allows to analyze the evolution of a database over its lifetime more precisely. In contrast to DAHLIA, our approach utilizes dynamic and static analysis to obtain a live visualization of the database and executed database queries from associated applications. Additionally, our approach addresses operators instead of developers. NakeDB [17] represents a dynamic visualization tool for huge databases. The tool generates database schema visualizations, more precisely visual graphs with color coded objects and shapes, based on database dumps. It features dynamic searching and filtering techniques, offers several visualization options, and provides interaction capabilities, like zooming and panning. In contrast to NakeDB, our approach enables a live visualization of databases, shows executed queries from applications, and provides an overview of the software landscape.

5 Conclusions

In this paper, we presented our RACCOON approach, which (i) enables the monitoring of applications and related databases and (ii) provides two, complementary visualization options based on the collected information. First, we offer a landscape-level perspective, based upon UML deployment and data-flow diagrams, which provides an overview of monitored applications and related databases. Second, we provide an *entity-relationship-diagram* inspired and *3D city metaphor*-based database-level perspective, which reveals database tables and relationships within a single database. It allows the operator to investigate the actual usage by applications. Both visualizations share the goal to provide a visual abstraction of the monitored objects. Once the operator requires more information, the respective visualization reveals additional objects or data in order to support his current task. Thus, we want to prevent overwhelming the user, when he employs our approach for the first time. Our open questions are:

- Which layout is suitable for our landscape-level perspective, which represents the applications and databases within a large software landscape?
- How do we link databases and related artifacts (e.g. deployed software) for our landscape-level perspective?
- Which advantage does our database-level perspective offer over traditional database diagrams like the entity relationship diagrams?
- To what extent is our approach also applicable for NoSQL databases?
- How can we successfully combine our database monitoring and visualization approach with existing Application Performance Management tools?
- Which related approaches or tools could be employed, when evaluating our approach within a controlled experiment?

References

[1] C. Zirkelbach, W. Hasselbring, and L. Carr, "Combining Kieker with Gephi for Performance Analysis and Interactive Trace Visualization," in *Proceedings of the Symposium on Software Performance 2015: Joint Developer and Community Meeting of Descartes/Kieker/Palladio*, 2015.

[2] OMG, *Unified Modeling Language*, Version 2.5, 2015.

[3] F. Fittkau, A. Krause, and W. Hasselbring, "Software Landscape and Application Visualization for System Comprehension with ExplorViz," *Information and Software Technology*, vol. 87, pp. 259–277, Jul. 2017. DOI: http://dx.doi.org/10.1016/j.infsof.2016.07.004.

[4] P. P.-S. Chen, "The Entity-Relationship Model – Toward a Unified View of Data," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, Mar. 1976, ISSN: 0362-5915. DOI: 10.1145/320434.320440.

[5] C. Zirkelbach, "Juggling with Data: On the Lack of Database Monitoring in Long-Living Software Systems," in *Proceedings of the 4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems*, Hanover, Germany, 2017.

[6] R. Wettel and M. Lanza, "Visualizing Software Systems as Cities," in *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2007. DOI: 10.1109/VISSOF.2007.4290706.

[7] B. Jenny and N. Kelso, "Color Design for the Color Vision Impaired," *Cartographic Perspectives*, vol. 0, no. 58, 2007.

[8] A. Kuhn, P. Loretan, and O. Nierstrasz, "Consistent Layout for Thematic Software Maps," in *Proceedings of the WCRE*, 2008. DOI: 10.1109/WCRE.2008.45.

[9] L. K. Klauske, C. D. Schulze, M. Spönemann, and R. von Hanxleden, "Improved Layout for Data Flow Diagrams with Port Constraints," in *Diagrammatic Representation and Inference: 7th International Conference, Diagrams 2012, Canterbury, UK, July 2-6, 2012. Proceedings*, P. Cox, B. Plimmer, and P. Rodgers, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 65–79, ISBN: 978-3-642-31223-6. DOI: 10.1007/978-3-642-31223-6_11.

[10] T. H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Finding and Evaluating the Performance Impact of Redundant Data Access for Applications that are Developed Using Object-Relational Mapping Frameworks," *IEEE Transactions on Software Engineering*, vol. 42, no. 12, pp. 1148–1161, Dec. 2016, ISSN: 0098-5589. DOI: 10.1109/TSE.2016.2553039.

[11] S. Ray, B. Simion, and A. D. Brown, "Jackpine: A benchmark to evaluate spatial database performance," in *Proceedings of the 27th International Conference on Data Engineering*, 2011. DOI: 10.1109/ICDE.2011.5767929.

[12] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring, "Live trace visualization for comprehending large software landscapes: The ExplorViz approach," in *1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)*, Sep. 2013. DOI: 10.1109/VISSOFT.2013.6650536.

[13] F. Fittkau, S. Roth, and W. Hasselbring, "ExplorViz: Visual runtime behavior analysis of enterprise application landscapes," in *23rd European Conference on Information Systems (ECIS 2015 Completed Research Papers)*, AIS Electronic Library, May 2015, pp. 1–13. DOI: 10.18151/7217313.

[14] J. Waller, C. Wulf, F. Fittkau, P. Döhring, and W. Hasselbring, "SynchroVis: 3D visualization of monitoring traces in the city metaphor for analyzing concurrency," in *1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)*, Sep. 2013. DOI: 10.1109/VISSOFT.2013.6650520.

[15] L. Meurice and A. Cleve, "DAHLIA: A Visual Analyzer of Database Schema Evolution," in *Proceedings of the CSMR-WCRE*, 2014.

[16] ——, "DAHLIA 2.0: A Visual Analyzer of Database Usage in Dynamic and Heterogeneous Systems," in *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT)*, 2016.

[17] L. M. Cortes-Pena, Y. Han, N. Pradhan, and R. Rigaux, *NakeDB: Database Schema Visualization*, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.460.5562, 2008.