# Kieker4DQL: Declarative Performance Measurement

Matthias Blohm
University of Stuttgart
Stuttgart, Germany
st140250@stud.uni-stuttgart.de

Sebastian Vogel
University of Stuttgart
Stuttgart, Germany
st140376@stud.uni-stuttgart.de

Maksim Pahlberg
University of Stuttgart
Stuttgart, Germany
st140177@stud.uni-stuttgart.de

Jürgen Walter
University of Würzburg
Würzburg, Germany
juergen.walter@uni-wuerzburg.de

Dušan Okanović
University of Stuttgart
Stuttgart, Germany
okanovic@informatik.uni-stuttgart.de

## Abstract

The Descartes Query Language (DQL) enables to query the performance of a system using adapters to various solution approaches. Thereby, DQL is a realization of the vision of Declarative Performance Engineering (DPE) which decouples the description of user concerns (performance questions and goals) from the task of selecting and applying a specific solution approach. While model-based approaches are already supported by DQL, measurement based approaches are not. In this paper we present the Kieker4DQL—an adapter for DQL that performs query processing for Application Monitoring Tools, the Kieker in particular. It creates a tailored monitoring configuration to start measurements. The resulting monitoring data is filtered and interpreted according to the query.

## 1 Introduction

Performance analysis approaches can be distinguished into model-based and measurement-based approaches. Model-based approaches, e.g. DML, PCM, analyze a model of a system in order to draw conclusions about the performance. On the other hand, measurement-based approaches and tools, e.g. Kieker, DynaTrace, inspectIT, draw these conclusions based on the data obtained by performing measurements in a live system. There is a huge need for a unified view on performance [1, 5], but there is no tool that provides such a unified view, although there are some approaches in development [3].

Although there are approaches to extract performance models from Application Performance Monitoring (APM) data, we still see a gap between measurement-based and model-based analysis. Descartes Query Language (DQL) framework aims to bridge this gap by providing an interface that abstracts performance analysis from the underlying approach, as described in [5]. While there are many applications scenarios having suitable model-based solution strategies, there are situations where it is better to apply measurement-based analysis.

- **SLA surveillance** Service Level Agreements (SLA) have to be monitored as model-based approaches do not provide proof of SLA violations.
- **Reactive auto-scaling**. In scenarios where load forecasting is difficult, model-based approaches introduce inaccuracy.
- **Live dashboard** Since models are abstractions, they may not depict the current state of the monitored system.

In all application scenarios, performance engineers are interested in performance metrics of a software system. Examples of these concerns are the response time of a `service1` or the utilization of a `CPU1`. Resulting questions could be, e.g., *What is the response time of service1?* or *What is the utilization of CPU1?*.

Performance engineers can use the DQL to define these concerns. While there are various implementations for model-based approaches, the DQL does not yet support measurement-based approach. In order to enable the DQL to connect to a measurement-based approach, we have to implement an adapter. Adapters control any monitoring or processing required by the underlying performance evaluation approaches.

This paper presents an adapter for translating performance concerns formulated using the DQL to a measurement-based approach—the Kieker Monitoring Framework [2]. It is used to create the configuration the Kieker will use for monitoring, and to extract the data from the Kieker's Monitoring Logs/Streams.

The rest of the paper is organized as follows. Section 2 presents a sketch of the process of answering queries. Section 3 explains the adapter design. Section 4 evaluates the proof of concept implementation, by analyzing the data obtained by running the Kieker with the generated monitoring configuration. Finally, Section 5 draws conclusions and outlines the future work. Runnable examples from the paper are available online.[1]

[1] http://dx.doi.org/10.5281/zenodo.61281


Figure 1: Query answering process using Kieker APM and Kieker4DQL adapter

## 2 DQL Query Answering Process

Once the performance concern has been declared, the DQL tooling executes the query by translating it via an adapter to the language of the underlying tool, which is used to gather data concerning performance. After the tool sends the data back to DQL, this data is used to answer the query.

Figure 1 shows the steps in processing of the DQL query, when the Kieker tool is used. Using DQL, the user will specify the query. This query is then analyzed by the adapter (step 1) in order to generate Kieker configuration tailored to the query (step 2). This configuration enables only the monitoring that is relevant for answering of the query, which reduces the monitoring overhead. Obtained data is than, using Kieker's filters (step 3), sent back to the DQL to present the results (step 4).

## 3 Adapter Design

The adapter architecture, displayed in Figure 2, is divided into two separate modules: the *Configuration Generation Module* and the *Filter Module*.

**Configuration Generation Module** This module is responsible for generating the Kieker configuration files, tailored to DQL queries. In the process of generation, the query is analyzed and the points of interest are identified in the monitored software. Only these points in the software will be instrumented.

The resulting configuration is stored in the *aop.xml* file, which is made available to Kieker. For example, if the DQL query requires the monitoring of the `CatalogService`, the monitoring will be as shown in the following listing:

```
1 <aspectj>
2   <weaver options="">
3     <include within="...␣.CatalogService"/>
4   </weaver>
5   <aspects>
6     <aspect name="...␣.OperationExecutionAspectFull"/>
7   </aspects>
8 </aspectj>
```

Listing 1: Example monitoring configuration

It includes what is going to be monitored (line 3), and with which monitoring probe (line 6).

**Filtering Module** This module reads the data from Kieker Monitoring Logs, using the Kieker Analysis component. It uses the data obtained using the configuration generated with *Configuration-Generation-Module*, but it can also work with the data generated using the Kieker default configuration. This can be useful if the monitored system does not allow the instrumentation to change. The module performs pre-processing of the data using filters.

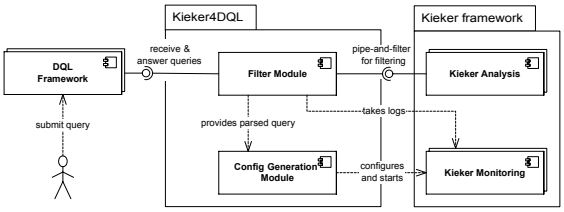


Figure 2: Kieker4DQL Adapter Architecture

We employ the Kieker analysis framework and its extensible pipe-and-filter architecture.The module is essentially a set of interconnected filters that can filter out the results according to a given DQL query. It converts the resulting monitoring data to DQL data structures.

In the implementation of this module, we reused the Performance Model Extractor (PMX) [2] which already provides mechanisms and data structures for the Kieker data analysis.

## 4 Evaluation

We have two main evaluation goals for our proof of concept implementation. We want to:

- evaluate if the implementation works as intended
- evaluate measurement overhead of the tailored monitoring configuration.

**Evaluation setup** To evaluate the functionality of our proof of concept, we executed two queries and generated the configuration files. These files were used to run the monitoring of the sample application. Results are than filtered and presented using the DQL plugin for Eclipse.

To test the overhead, we performed the testing of the adapter in two separate configurations:

- with the default Kieker configuration, that allows monitoring of the whole system,
- with the tailored configuration for the provided queries.

The evaluation was performed using JPetStore application that is available with the Kieker distribution. All tests were performed using the bundled load generator, and were run for 200 seconds. They were conducted on a notebook with Intel Core i5-4310M, 8 $GB$ RAM, SSD and Windows 10 running Xubuntu 15.10 in a VM.

**Results and Discussion** To test if the implementation works as intended, we executed two queries, shown in listings 2 and 3. The first query retrieves

[2] www.descartes.tools/pmx

```
SELECT res1.utilization
FOR RESOURCE 'cpu1' AS res1
USING kieker@'dql.properties';
```

Listing 2: Query for the utilization of the *CPU1*

```
SELECT srv1.responseTime
FOR SERVICE 'CatalogActionBean.getItem()' AS srv1
USING kieker@'dql.properties';
```

Listing 3: Query for the average response time of the service

the resource utilization, in this case the utilization of the CPU1. The second query retrieves the response times of the `CatalogService`.

The resulting plot for the first query, shown in Figure 3, shows that utilization is high at the start of the experiment, until steady state is reached. For the response times, only the plot for a method `getItem()` in JPetStore's `CatalogService` class is shown (Figure 4). The plot shows various peaks in response times, and for further analysis these peaks should be removed from the data using, e.g., outlier detection techniques. Plots for response times of other methods are similar.

**Performance/Overhead** While the query execution results show no difference whether we use tailored monitoring configuration or monitoring of the whole application, the performance overhead changes significantly. As expected, monitoring using tailored configuration provides lower overhead, than monitoring of the whole application. On our test system (Core i5, 8 *GB* RAM, Xubuntu 15.10 in a VM), response times using tailored monitoring configuration are around 16% faster.

## 5 Conclusion

In this paper we presented the Kieker4DQL, an adapter that allows the DQL to use the data from the Kieker APM. While DQL already has wide support for model-based approaches, this adapter will allow the use of the measured data from the live system. Our adapter enables to query performance measurements captured using the Kieker APM and might serve as a model for the implementation of further measurement-based adapters.

For future work we aim to extend this support to more APM tools. One way of doing this is through the use of OPEN.xtrace format [4]. In addition, DQL could be equipped with mechanisms to extract performance models using APM data, e.g., Performance Model Extractor (PMX) as it also builds upon the Kieker APM data format.

## Acknowledgments

This work is supported by the German Research Foundation (DFG) in the Priority Programme "DFG-SPP 1593: Design For Future—Managed Software Evolution" (HO 5721/1-1 and KO 3445/15-1) and by the Research Group of the Standard Performance Evaluation Corporation (SPEC).

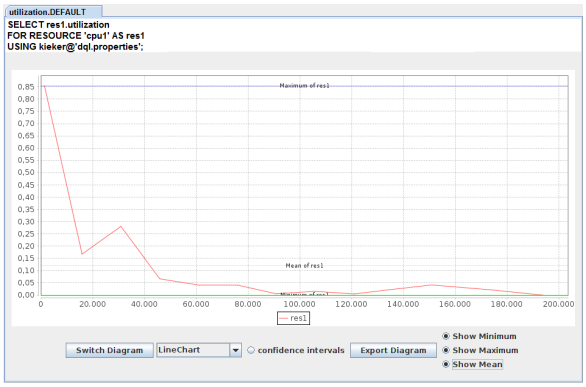


Figure 3: The execution of the query in Listing 2

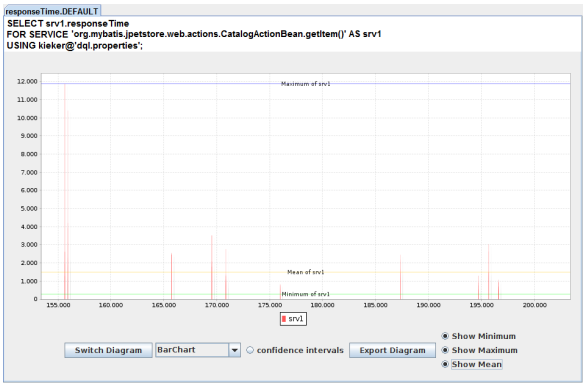


Figure 4: The execution of the query in Listing 3

## References

[1] M. Woodside, G. Franks, and D. C. Petriu. "The Future of Software Performance Engineering". In: *2007 Future of Software Engineering*. FOSE '07. IEEE Computer Society, 2007, pp. 171–187.

[2] A. van Hoorn, J. Waller, and W. Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *Proc. 3rd ACM/SPEC Int. Conf. on Performance Eng. (ICPE '12)*. 2012, pp. 247–248.

[3] R. Heinrich et al. "Integrating Run-Time Observations and Design Component Models for Cloud System Analysis". In: *Proc. 9th Workshop on Models@run.time*. Vol. 1270. 2014, pp. 41–46.

[4] D. Okanović et al. "Towards Performance Tooling Interoperability: An Open Format for Representing Execution Traces". In: *Proceedings of the 13th European Workshop on Performance Engineering (EPEW '16)*. Chios, Greece, 2016.

[5] J. Walter et al. "Asking "What?", Automating the "How?": The Vision of Declarative Performance Engineering". In: *Proc. 7th ACM/SPEC Int. Conf. on Performance Eng. (ICPE 2016)*. 2016, pp. 91–94.