Controlling the Palladio Bench using the Descartes Query Language

Fabian Gorsler, Fabian Brosig, Samuel Kounev

Descartes Research Group
Karlsruhe Institute of Technology (KIT)
Am Fasanengarten 5
D-76131 Karlsruhe, Germany
gorsler@ira.uka.de
{brosig, kounev}@kit.edu

Abstract: The Palladio Bench is a tool to model, simulate and analyze Palladio Component Model (PCM) instances. However, for the Palladio Bench, no single interface to automate experiments or Application Programming Interface (API) to trigger the simulation of PCM instances and to extract performance prediction results is available. The Descartes Query Language (DQL) is a novel approach of a declarative query language to integrate different performance modeling and prediction techniques behind a unifying interface. Users benefit from the abstraction of specific tools to prepare and trigger performance predictions, less effort to obtain performance metrics of interest, and means to automate performance predictions. In this paper, we describe the realization of a DQL Connector for PCM and demonstrate the applicability of our approach in a case study.

1 Introduction

Performance predictions are normally subject to a recurring process. The process in Figure 1 is an example of this process. First, users need to specify which performance metrics they demand to trigger a performance prediction using tools compatible to their performance modeling formalism. Second, the architecture-level performance model needs to be transformed into an *analysis model* to perform the performance prediction. Analysis models are usually more abstract and capture only the high-level details of an architecture-level performance model. Examples of architecture-level performance models include the Descartes Meta-Model (DMM) [KBH12] and the Palladio Component Model (PCM) [RBB+11, BKR09]. We assume an architecture-level performance model is already available. Third, after the results of a performance prediction are available, the user can extract the relevant performance metrics. Typically, this process is executed in iterations and involves the use of one or more tools for each step to achieve the demanded goals. Thus, the manual effort is high and users need to learn the low-level details of each tool that is part of the performance prediction process.

The PCM is an architecture-level model performance modeling language to model soft-

Proc. Kieker/Palladio Days 2013, Nov. 27–29, Karlsruhe, Germany
Available online: http://ceur-ws.org/Vol-1083/

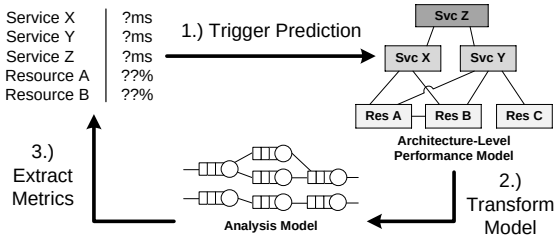

Figure 1: The performance prediction process

ware systems based on component-oriented architectures for performance predictions already at design time. The Palladio Bench provides the necessary tools to model PCM instances, to trigger performance predictions and to extract the results. However, the Palladio Bench does not provide the means to automate the steps in the performance prediction process and provides no single Application Programming Interface (API) for automation. Each step involves one or multiple actions necessary to trigger the transition to the next step. In this paper, we present the Descartes Query Language (DQL), contribute a DQL Connector for PCM and show the applicability with PCM in a case study. DQL is our novel approach of a declarative query language to ease performance predictions and to automate recurring tasks [Gor13]. With DQL, different performance modeling formalisms and prediction techniques can be integrated as DQL Connectors with a single unified interface and API. Users benefit from less manual effort, a flat learning curve to trigger performance predictions, a declarative description to obtain results and an API to embed performance predictions in software components.

The remainder of this paper is structured as follows: Section 2 introduces DQL and relevant approaches. In Section 3 the implementation of a DQL Connector for PCM will be described and Section 4 demonstrates the applicability of our approach. Section 5 summarizes and discusses our work and the results presented and gives an outlook to future work.

2 Foundations and Related Work

2.1 Descartes Query Language

The Descartes Query Language (DQL) is our novel approach of a query language to trigger performance predictions independent of a specific performance modeling formalism or prediction technique [Gor13]. DQL aims to integrate existing performance prediction approaches to unify their interfaces by a declarative language. The design DQL focuses on architecture-level performance models, e.g. the Descartes Meta-Model (DMM) or Palladio Component Model (PCM), and captures usage scenarios that can be either *offline scenarios* during the design time of software systems or *online scenarios* during the run-time of a

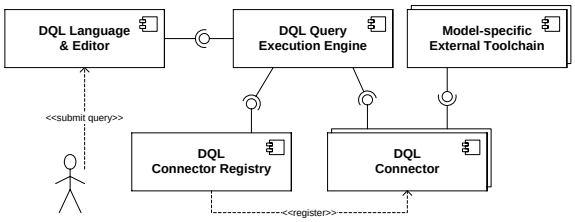

Figure 2: Architecture of DQL

software system [KBH12, RBB+11]. In this paper, we focus on *Model Structure Queries* to obtain structural information of a performance model and *Performance Metrics Queries* to trigger performance predictions and to extract result. Furthermore, DQL automates performance predictions through the exploration of Degrees of Freedom (DoFs). DoFs express the configuration parameter space of performance model entities to find solutions for optimization problems automatically [KAM13, HvHK+13].

Figure 2 shows the architecture of DQL. The architecture consists of three major parts: (i) *DQL Language & Editor* provides an Xtext[1]-based editor, query parser and Application Programming Interface (API) to DQL. This part of DQL receives queries from external sources, either through the editor part from users or through the API part from software components, and delegates them to the DQL Query Execution Engine (QEE). In (ii) the *DQL QEE* encapsulates the core logic of DQL and performs the performance modeling formalism-independent processing of queries. Finally, in (iii) *DQL Connectors* encapsulate all performance modeling formalism-specific logic and control the execution of performance predictions. The *DQL Connector Registry* is a utility component to manage available DQL Connectors in a DQL environment.

In Fig. 3 the DQL Editor embedded in the Eclipse Integrated Development Environment (IDE) is shown. The upper half contains a DQL query that is part of the case study in Sec. 4, while the lower half visualizes the result of the query.

2.2 Related Approaches from SPE

In Software Performance Engineering (SPE) the main objective is the systematic analysis of the performance of software systems [SS91, MDA04]. Many approaches for the analysis of performance models have emerged that differ in their expressiveness, computing effort and modeling formalism [Koz10, BDIS04]. Due to the different interfaces and applications of tools, new problems arise. In [WFP07] the problems are summarized: *"As a result no tool does the job the user needs, so the user goes and invents one. Further, various tools all have different forms of output which makes interoperability challenging at best."*. We position our approach between approaches for *intermediate modeling* and *metric modeling*.

[1] `http://www.eclipse.org/Xtext/`

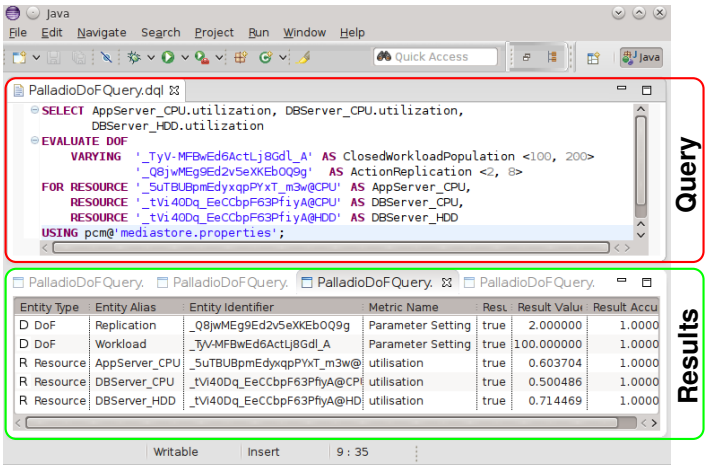

Figure 3: DQL Query Editor and Result View

Intermediate modeling techniques such as KLAPER and CSM reduce the effort for the transformation of performance models between different representations and purposes [GMRS08, WPP+05]. Unfortunately, these approaches expose no unified interfaces, tools lack from missing APIs and online prediction scenarios during run-time of systems are not treated. Thus, we propose to implement intermediate modeling techniques as DQL Connectors and to make use of their existing transformations.

SMM is a standard for the modeling of metrics and MAMBA is an implementation of SMM [Obj12, FvHJ+11, FvHJ+12]. The modeling of metrics allows the standardized analysis of metrics independent of their source and eases the development tools. However, the Measurement Architecture for Model-Based Analysis (MAMBA) approach is not intended to control a performance prediction process and is limited to the access to performance metrics. We propose the integration of MAMBA and DQL to form a unified interface to performance data repositories and performance prediction techniques through a DQL Connector for MAMBA.

3 Implementation

3.1 Mapping from PCM to DQL

As first step towards the realization of a DQL Connector for PCM, we describe our mapping approach from PCM to the Mapping Meta-Model. The Mapping Meta-Model as shown in Figure 4 serves as abstraction layer to integrate performance modeling formalisms into DQL. Furthermore, instances of the Mapping Meta-Model serve to trigger performance predictions in requests from the DQL QEE to DQL Connectors and to return

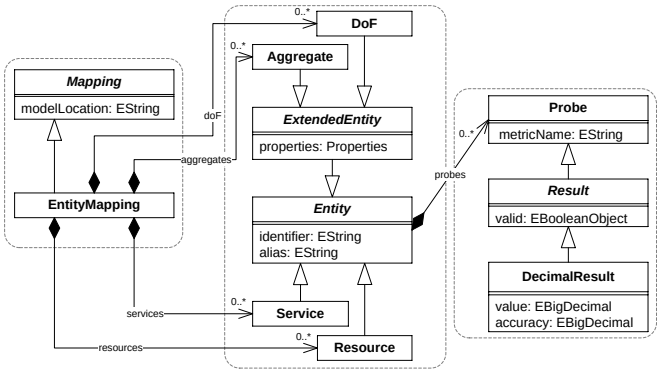

Figure 4: The Mapping Meta-Model

results from DQL Connectors to the DQL QEE and users. The Mapping Meta-Model describes essential model entities of architecture-level performance models, e.g. `Resource` and `Service`. In a request to obtain performance metrics for such entities, instances of the type `Probe` attach to `Resources` and `Services` and contain a reference to a metric of interest. On the other hand, `Probes` are replaced by instances of types derived from `Result` to represent responses. The Mapping Meta-Model contains additional types to reference DoFs and to store statistical aggregates computed on top of plain performance metrics.

In the *Usage Model* of PCM, workload-specific modeling aspects are modeled by domain experts to represent usage scenarios of the modeled software system [RBB+11]. We identify the types `UsageScenario` and `EntryLevelSystemCall` as relevant entities in the PCM Usage Model to be mapped to the type `Service` in the Mapping Meta-Model. In addition, PCM provides a *Repository Model* to model and store software components with their behavioral descriptions [RBB+11]. Here, the type `ExternalCallAction` is relevant and mapped to the `Service` type of the Mapping Meta-Model. `ExternalCallAction` represents the call of a service provided by another component. Finally, in the *Resource Environment Model* of PCM the types `ProcessingResourceSpecification` and `CommunicationLinkResourceSpecification` can be mapped to the Mapping Meta-Model type `Resource`. The first type represents active resources of a hardware server, the latter type represents the network link of a hardware server. With this mapping, the following performance metrics can be extracted from PCM simulations: (i) *demandedTime* and *utilization* for `Resource` and (ii) *responseTime* for `Service` [Mer11].

For the mapping of `Resources` we employ a workaround. The workaround is necessary due to a shortcoming of the the SensorFramework and the format of results. To obtain result results for specific resources, we modify the identifiers with the suffixes `@CPU` and `@HDD` in case of a `ProcessingResourceSpecification` and `@LAN` in case of a `CommunicationLinkResourceSpecification`. By this workaround, the DQL Connector can access the computed results through the SensorFramework and provide a distinct mapping to trigger predictions and to obtain results.

The DQL Connector for PCM supports DoFs in PCM instances and varies them using

the PCM Experiment Automation API [Mer11]. A DoF in a PCM instance can be, e.g., the workload population in the PCM Usage Model to analyze the behavior of a software system under varying workload intensities. We implement DoFs based on the available variations for PCM types and support the full exploration of the resulting configuration parameter space. The full parameter space spans across $\Pi_{i=1}^{n} d_i$ different combinations, where n is the amount of available DoFs and d_i is the number of different settings of a DoF i. For each combination of DoFs, the DQL Connector triggers one PCM simulation and returns one result set.

3.2 DQL Connector for PCM

The initial steps to develop the DQL Connector for PCM are the creation of a new OSGi Bundle, to enclose meta-information in the OSGi Bundle, and to provide a OSGi Declarative Service implementing `ConnectorProvider` to the OSGi run-time [OSG11, OSG12]. `ConnectorProvider` is an interface to implement a factory class that is used to create instaces of the classes derived from `QueryConnector`. The implementation of `QueryConnectors` are the subsequent steps to realize the DQL Connector and to enrich the functionality.

The DQL Connector for PCM consists of two implementations of `QueryConnector`. First, the `ModelStructureQueryConnector` is used to obtain strutural information from a performance model. The implementation is based on Eclipse Modeling Framework (EMF) operations and Object Constraint Language (OCL) to search for instances of the types described in the previous section. The available performance metrics for the eligible entities are mapped in the DQL Connector. Second, the `PerformanceMetricsQueryConnector` configures and controls the performance prediction process and extracts the performance metrics of interest. To configure and control performance predictions, the DQL Connector for PCM relies on the PCM Experiment Automation API [Mer11]. The extraction of performance metrics is realized through direct access to the SensorFramework API.

4 Case Study

4.1 Description of the MediaStore Example

The MediaStore[2] is a running example for applications of PCM and demonstrates the features of the Palladio Bench [BKR09]. It is an instance of the PCM with a component-oriented architecture and can be used to trigger simulations that derive performance metrics. The architecture is built up on a three tier approach and made up of (i) a web front-end component for users, (ii) a business tier consisting of the core business logic and a digital watermarking component, and (iii) a database tier. Tiers (i) and (ii) are deployed on an ap-

[2] `https://sdqweb.ipd.kit.edu/wiki/PCM_MediaStore_Example_Workspace`

```
LIST ENTITIES
USING pcm@'mediastore.properties';
```

Listing 1: List Entities Query

```
LIST METRICS (RESOURCE '_5uTBUBpmEdyxqpPYxT_m3w@CPU' AS AppServer_CPU,
  RESOURCE '_tVi40Dq_EeCCbpF63PfiyA@CPU' AS DBServer_CPU)
  RESOURCE '_tVi40Dq_EeCCbpF63PfiyA@HDD' AS DBServer_HDD)
USING pcm@'mediastore.properties';
```

Listing 2: List Metrics Query

plication server (*AppServer*), tier (iii) is deployed on a database server (*DBServer*). Both are interconnected by a network link.

4.2 Performance Analysis of the MediaStore Example

This section is presents an outline how to conduct a performance analysis of a PCM model through the DQL Connector for PCM. The performance analysis relies only on DQL queries and omits the Palladio Bench as interface. The case study consists of three main objectives: (i) To explore the model structure of the MediaStore example, (ii) to analyze available performance metrics for specific model entities, and (iii) to trigger a performance prediction and extract performance metrics. Additionally, we present an example how to automate experiment series in the Palladio Bench using a single DQL query.

The first objective in this case study is to obtain a listing of all available performance-relevant entities in the MediaStore. The result set contains all entities, i.e. all mapped `Resources` and `Services`, and their absolute identifiers. Listing 1 shows the corresponding query from the Model Structure Query Class. The second line in the query contains the `USING` keyword. It provides means to (i) select an adequate DQL Connector to execute the query and (ii) it contains the model location. The interpretation of the model location is subject to the referenced DQL Connector. In case of PCM, a properties file with references to all required PCM sub-models is used. The first line of the query contains the `LIST ENTITIES` expression to obtain all model entities and to interpret the DQL mapping of the entities found.

The second objective is to obtain the available performance metrics for the model entities of interest. The query in Listing 2 contains the corresponding example and is again part of the Model Structure Query Class. Here, the `LIST METRICS` expression is used together with a set of references of relevant model entities. The references to model entities contain the DQL type, the performance model formalism-specific absolute identifier and an optional alias to identify the reference in result sets easily. The DQL Editor provides the necessary means for the auto-completion of absolute identifiers of resources and services. The result set of this query contains for each of the requested resources the available metrics

```
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization, DBServer_HDD.utilization
FOR RESOURCE '_5uTBUBpmEdyxqpPYxT_m3w@CPU' AS AppServer_CPU,
    RESOURCE '_tVi40Dq_EeCCbpF63PfiyA@CPU' AS DBServer_CPU,
    RESOURCE '_tVi40Dq_EeCCbpF63PfiyA@HDD' AS DBServer_HDD
USING pcm@'mediastore.properties';
```

Listing 3: Basic Query in the Palladio Bench

demandedTime and *utilization* as described in the preceding section.

Finally, the third objective is to trigger a performance prediction and to extract the performance metrics of interest from the performance prediction results. The query in Listing 3 contains the necessary statement for this objective and is from the Performance Metrics Query Class. The `FOR` expression references the model entities of interest. A DQL Connector may use this information to provide tailored predictions, which is, in case of the PCM Experiment Automation API, not applicable, but still necessary for the `SELECT` expression. Only referenced model entities can be used in the `SELECT` expression. The `SELECT` expression contains all performance metrics of interest for specific model entities. In this case, the *utilization rates* for the entities *AppServer_CPU*, *DBServer_CPU* and *DBServer_HDD* are of interest. This query triggers a performance prediction through a simulation run in the Palladio Bench and the performance metrics are returned as mean values computed directly from the results available through the SensorFramework.

Additionally, as an example for automation of tasks in the Palladio Bench, Listing 4 contains a DoF Query. The query extends Listing 3 through the `EVALUATE DOF` expression. The `EVALUATE DOF` expression together with the `VARYING` expression triggers an full exploration of the specified DoFs. Here, the exploration varies two parameters with two settings each, leading to a total amount of $2 \times 2 = 4$ independent simulations in the Palladio Bench and the same amount of result sets presented to the user. The exploration varies (i) the workload intensity of the closed workload used in the MediaStore example and (ii) changes the component-internal behavior through the replication of a given internal action. For both variations, the PCM Experiment Automation API provides implementations to vary the PCM model instances.

As final part of the case study, Figure 3 shows the user interface of DQL. The DQL Editor, shown in the upper half, provides means to edit queries with completion features DQL expressions, context-sensitive identifiers of model entities, and syntax highlighting. The result visualization, shown in the lower half, is a tabular representation of a Mapping Meta-Model instances and displays all types referenced by the `EntityMapping`, see Figure 4 and a visualization of the Mapping Meta-Model returned as result in the lower half. The screenshot shows the state of the Eclipse-based user interface after the execution of the query from Listing 4. Thus, four results are shown in the lower half that can be analyzed by a user.

```
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization, DBServer_HDD.utilization
EVALUATE DOF
  VARYING '_TyV-MFBwEd6ActLj8Gdl_A' AS ClosedWorkloadPopulation <100, 200>
          '_Q8jwMEg9Ed2v5eXKEbOQ9g' AS ActionReplication <2, 8>
FOR RESOURCE '_5uTBUBpmEdyxqpPYxT_m3w@CPU' AS AppServer_CPU,
    RESOURCE '_tVi40Dq_EeCCbpF63PfiyA@CPU' AS DBServer_CPU,
    RESOURCE '_tVi40Dq_EeCCbpF63PfiyA@HDD' AS DBServer_HDD
USING pcm@'mediastore.properties';
```

Listing 4: Complex DoF Query in the Palladio Bench

5 Conclusion & Future Work

We presented the Descartes Query Language (DQL), a novel query language to specify performance queries, and a DQL Connector for Palladio Component Model (PCM). DQL unifies the interfaces of available performance modeling formalisms and their prediction techniques to provide a common Application Programming Interface (API). DQL is independent of the employed modeling formalisms, hides low-level details of performance prediction techniques and thus reduces the manual effort and the learning curve when working with performance models.

Instead, the DQL Connector for PCM encapsulates all PCM-specific details and unifies the APIs of Eclipse Modeling Framework (EMF), Object Constraint Language (OCL), PCM Experiment Automation and the SensorFramework. Our case study with the MediaStore example showed the applicability of our approach to control the Palladio Bench through DQL. The current state of the implementation can be used to conduct a performance analysis and extends the functionality of the Palladio Bench by means to automate performance predictions with Degrees of Freedom (DoFs).

As part of our future work on DQL, we plan to integrate further DQL Connectors and to provide additional query classes. Additional query classes in DQL are intended to address problems like the automated detection of bottlenecks. These classes are intended as a step towards *goal-oriented queries*.

References

[BDIS04] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 2004.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 2009.

[FvHJ+11] Sören Frey, A van Hoorn, R Jung, Wilhelm Hasselbring, and Benjamin Kiel. MAMBA: A measurement architecture for model-based analysis. Technical report, Department of Computer Science, University of Kiel, 2011.

[FvHJ+12] Sören Frey, A van Hoorn, R Jung, Benjamin Kiel, and Wilhelm Hasselbring. MAMBA: Model-Based Software Analysis Utilizing OMG's SMM. In *14. Workshop Software-Reengineering*, 2012.

[GMRS08] Vincenzo Grassi, Raffaela Mirandola, Enrico Randazzo, and Antonino Sabetta. KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability. volume 5153 of *Lecture Notes in Computer Science*. 2008.

[Gor13] Fabian Gorsler. Online Performance Queries for Architecture-Level Performance Models. Master's thesis, Karlsruhe Institute of Technology (KIT), 2013.

[HvHK+13] Nikolaus Huber, André van Hoorn, Anne Koziolek, Fabian Brosig, and Samuel Kounev. Modeling Run-Time Adaptation at the System Architecture Level in Dynamic Service-Oriented Environments. *Service Oriented Computing and Applications Journal (SOCA)*, 2013.

[KAM13] Anne Koziolek, Danilo Ardagna, and Raffaela Mirandola. Hybrid Multi-Attribute QoS Optimization in Component Based Software Systems. *Journal of Systems and Software*, 86(10), 2013.

[KBH12] Samuel Kounev, Fabian Brosig, and Nikolaus Huber. The Descartes Meta-Model (DMM). Technical report, Karlsruhe Institute of Technology (KIT), Karlsruhe, 2012.

[Koz10] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 2010.

[MDA04] Daniel A. Menasce, Lawrence W. Dowdy, and Virgilio A. F. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[Mer11] Philipp Merkle. *Comparing Process- and Event-oriented Software Performance Simulation*. Master thesis, Karlsruhe Institute of Technology (KIT), 2011.

[Obj12] Object Management Group. Structured Metrics Metamodel 1.0, 2012.

[OSG11] OSGi Alliance. OSGi Service Platform Core Specification - Release 4, Version 4.3. 2011.

[OSG12] OSGi Alliance. OSGi Service Platform Service Compendium - Release 4, Version 4.3. 2012.

[RBB+11] Ralf Reussner, Steffen Becker, Erik Burger, Jens Happe, Michael Hauck, Anne Koziolek, Heiko Koziolek, Klaus Krogmann, and Michael Kuperberg. The Palladio Component Model. Technical report, Karlsruhe Institute of Technology (KIT), Karlsruhe, 2011.

[SS91] John Smith and Connie U. Smith. Performance Engineering of Software Systems. *The Journal of the Operational Research Society*, 1991.

[WFP07] Murray Woodside, Greg Franks, and Dorina C. Petriu. The Future of Software Performance Engineering. In *Future of Software Engineering (FOSE '07)*. IEEE, 2007.

[WPP+05] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Toqeer Israr, and Jose Merseguer. Performance by unified model analysis (PUMA). In *WOSP '05*. ACM Press, 2005.