Controlling the Palladio Bench using the Descartes Query Language

Kieker/Palladio Days 2013
Fabian Gorsler, Fabian Brosig, Samuel Kounev | 2013-11-27
<table>
<thead>
<tr>
<th>Service</th>
<th>?ms</th>
<th>Resource</th>
<th>??%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service X</td>
<td>?ms</td>
<td>Resource A</td>
<td>??%</td>
</tr>
<tr>
<td>Service Y</td>
<td>?ms</td>
<td>Resource B</td>
<td>??%</td>
</tr>
<tr>
<td>Service Z</td>
<td>?ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: The Performance Prediction Process
### Our Motivation

<table>
<thead>
<tr>
<th>Service X</th>
<th>?ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Y</td>
<td>?ms</td>
</tr>
<tr>
<td>Service Z</td>
<td>?ms</td>
</tr>
<tr>
<td>Resource A</td>
<td>??%</td>
</tr>
<tr>
<td>Resource B</td>
<td>??%</td>
</tr>
</tbody>
</table>

---

**Figure 1**: The Performance Prediction Process

---

**Introduction of DQL**

*Fabian Gorsler, Fabian Brosig, Samuel Kounev – Controlling the Palladio Bench using DQL*  
2013-11-27
Our Motivation

<table>
<thead>
<tr>
<th>Service</th>
<th>?ms</th>
<th>Resource</th>
<th>??%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service X</td>
<td></td>
<td>Resource A</td>
<td></td>
</tr>
<tr>
<td>Service Y</td>
<td></td>
<td>Resource B</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: The Performance Prediction Process
Our Motivation

<table>
<thead>
<tr>
<th>Service</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service X</td>
<td>3</td>
</tr>
<tr>
<td>Service Y</td>
<td>4</td>
</tr>
<tr>
<td>Service Z</td>
<td>6</td>
</tr>
<tr>
<td>Resource A</td>
<td>33%</td>
</tr>
<tr>
<td>Resource B</td>
<td>78%</td>
</tr>
</tbody>
</table>

Figure 1: The Performance Prediction Process
Our Motivation

<table>
<thead>
<tr>
<th>Service</th>
<th>Time (ms)</th>
<th>Resource</th>
<th>Usage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>3</td>
<td>A</td>
<td>33%</td>
</tr>
<tr>
<td>Y</td>
<td>4</td>
<td>A.2</td>
<td>78%</td>
</tr>
<tr>
<td>Z</td>
<td>6</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: The Performance Prediction Process
Our Approach

What is the Descartes Query Language (DQL)?

- A *declarative query language* [Gorsler, 2013]
- *Independent* of a specific modeling or prediction formalism
- An approach *to integrate* existing tools and techniques
- An interface *to unify* other approaches’ interfaces
- Built on top of an *extensible architecture*

...and what it is not?

- Neither an approach for performance predictions, ...
- ...nor a model transformation approach
MediaStore and Outline

MediaStore

- MediaStore is an example of a three-tier web application
- Demonstrates capabilities of Palladio Component Model (PCM)
- Contains an AppServer and a DBServer with CPUs and HDDs each

⇒ Is DQL usable in such a scenario?

Outline

1. Determine performance-relevant entities and metrics
2. Trigger a simulation and extract performance metrics
3. Trigger an experiment series through Degrees of Freedom (DoFs)
Finding Entities

- Obtain information about *all performance-relevant entities*

  ➝ Interpret the model instance
Finding Entities

- Obtain information about *all performance-relevant entities*

  ⇒ Interpret the model instance

```plaintext
LIST ENTITIES
USING pcm@'mediastore.properties';
```
Finding Entities

- Obtain information about *all performance-relevant entities*

  ➞ Interpret the model instance

**LIST ENTITIES**

**USING pcm@’mediastore.properties’;**

<table>
<thead>
<tr>
<th>Exemplary Result:</th>
<th>Type</th>
<th>Id</th>
<th>Alias</th>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Resource</td>
<td>id1</td>
<td>AppServer_CPU</td>
<td>./</td>
<td>./</td>
</tr>
</tbody>
</table>
Backgrounds: Type Mapping

Figure 2: PCM ← DQL based on [Reussner et al., 2011]

- Direct EMF/Ecore accesses
- Loosely-coupled using **identifier values**
Determining available Metrics

- Obtain information about *available performance metrics*

→ Interpret the referenced entities
Determining available Metrics

- Obtain information about *available performance metrics*

  → Interpret the referenced entities

```sql
LIST METRICS
  (RESOURCE 'id1' AS AppServer_CPU,
   RESOURCE 'id2' AS DBServer_CPU
   RESOURCE 'id3' AS DBServer_HDD)
USING pcm@'mediastore.properties';
```
Determining available Metrics

- Obtain information about *available performance metrics*
  
  ➞ Interpret the referenced entities

**LIST METRICS**

```
(RESOURCE 'id1' AS AppServer_CPU,
 RESOURCE 'id2' AS DBServer_CPU
 RESOURCE 'id3' AS DBServer_HDD)
USING pcm@'mediastore.properties';
```

**Exemplary Result:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Id</th>
<th>Alias</th>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>id1</td>
<td>AppServer_CPU</td>
<td>Utilization</td>
<td>./.</td>
</tr>
</tbody>
</table>
Triggering a Simulation

- Request performance metrics for model entities
  ⇝ Trigger a simulation run and extract results

```sql
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization,
       DBServer_HDD.utilization
FOR RESOURCE 'id1' AS AppServer_CPU,
     RESOURCE 'id2' AS DBServer_CPU,
     RESOURCE 'id3' AS DBServer_HDD
USING pcm@'mediastore.properties';
```

Exemplary Result:

<table>
<thead>
<tr>
<th>Type</th>
<th>Id</th>
<th>Alias</th>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>id1</td>
<td>AppServer_CPU</td>
<td>Utilization</td>
<td>0.3671</td>
</tr>
</tbody>
</table>
Triggering a Simulation

- Request performance metrics for model entities
  \[\Rightarrow\] Trigger a simulation run and extract results

```
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization, 
       DBServer_HDD.utilization
FOR RESOURCE 'id1' AS AppServer_CPU, 
     RESOURCE 'id2' AS DBServer_CPU, 
     RESOURCE 'id3' AS DBServer_HDD
USING pcm@'mediastore.properties';
```
Triggering a Simulation

- Request performance metrics for model entities

   ⇝ Trigger a simulation run and extract results

```
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization,
       DBServer_HDD.utilization
FOR RESOURCE 'id1' AS AppServer_CPU,
   RESOURCE 'id2' AS DBServer_CPU,
   RESOURCE 'id3' AS DBServer_HDD
USING pcm@'mediastore.properties';
```

Exemplary Result:

<table>
<thead>
<tr>
<th>Type</th>
<th>Id</th>
<th>Alias</th>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>id1</td>
<td>AppServer_CPU</td>
<td>Utilization</td>
<td>0.3671</td>
</tr>
</tbody>
</table>

Introduction of DQL

Case Study MediaStore

Fabian Gorsler, Fabian Brosig, Samuel Kounev – Controlling the Palladio Bench using DQL

2013-11-27
Bonuses: Aggregates in DQL

- Computation of aggregated performance metrics
- Functionality \textit{independent} of a DQL Connector
 Bonus: Aggregates in DQL

- Computation of aggregated performance metrics
- Functionality *independent* of a DQL Connector

```sql
SELECT MEAN(AppServer1_CPU.utilization, AppServer2_CPU.utilization) FOR RESOURCE 'idA1' AS AppServer1_CPU, RESOURCE 'idA2' AS AppServer2_CPU, USING pcm@'mediastore.properties';
```
Backgrounds: Trigger and Extraction

- *Palladio Component Model (PCM) Experiment Automation* to trigger simulations [Merkle, 2011]
  - Part of the PCM Incubator, not yet stable

- Static mapping of available performance metrics
  - Resource: *demanded time* and *utilization*
  - Service: *response time*

- Direct access to performance metrics through *SensorFramework*
  - Workaround for unique identification of Resource instances

- Evaluate DoFs, i.e. variability, in model instances
- Automate experiment series with a single query
  $\Rightarrow$ Alter the model instance and trigger $i \times j \times \ldots$ simulations

SELECT
AppServer_CPU.utilization, DBServer_CPU.utilization, DBServer_HDD.utilization
EVALUATE DOF
VARYING 'id4' AS ClosedWorkloadPopulation <100, 200>
'id5' AS ActionReplication <2, 8>
FOR RESOURCE 'id1' AS AppServer_CPU,
RESOURCE 'id2' AS DBServer_CPU,
RESOURCE 'id3' AS DBServer_HDD
USING pcm@'mediastore.properties';

- Evaluate DoFs, i.e. variability, in model instances
- Automate experiment series with a single query

\[ \Rightarrow \text{Alter the model instance and trigger } i \times j \times \ldots \text{ simulations} \]

```
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization,
       DBServer_HDD.utilization

FOR RESOURCE 'id1' AS AppServer_CPU,
    RESOURCE 'id2' AS DBServer_CPU,
    RESOURCE 'id3' AS DBServer_HDD
USING pcm@'mediastore.properties';
```

- Evaluate DoFs, i.e. variability, in model instances
- Automate experiment series with a single query

\[ \Rightarrow \text{Alter the model instance and trigger } i \times j \times \ldots \text{ simulations} \]

```sql
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization, DBServer_HDD.utilization
FOR RESOURCE 'id1' AS AppServer_CPU,
     RESOURCE 'id2' AS DBServer_CPU,
     RESOURCE 'id3' AS DBServer_HDD
USING pcm@'mediastore.properties';
```

- Evaluate DoFs, i.e. variability, in model instances
- Automate experiment series with a single query

\[ \Rightarrow \text{Alter the model instance and trigger } i \times j \times \ldots \text{ simulations} \]

```sql
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization,
       DBServer_HDD.utilization
EVALUATE DOF
  VARYING 'id4' AS ClosedWorkloadPopulation <100, 200>
       'id5' AS ActionReplication <2, 8>
FOR RESOURCE 'id1' AS AppServer_CPU,
    RESOURCE 'id2' AS DBServer_CPU,
    RESOURCE 'id3' AS DBServer_HDD
USING pcm@'mediastore.properties';
```
DQL Editor and Result Visualization

**Query**

```
SELECT AppServer_CPU.utilization, DBServer_CPU.utilization, DBServer_HDD.utilization
EVALUATE DOF
VARYING 'TyV-MFbW6d6ActLj8Gdl_A' AS ClosedWorkloadPopulation <100, 200>
'_Q8jweEg9Ed2v5eXKb0Q9g' AS ActionReplication <2, 8>
FORE RESOURSE '5uTBUBpmEdyxpPYxt_m3w@CPU' AS AppServer_CPU,
RESOURSE '_tvI4Dq_EeCcBp63PfiyA@CPU' AS DBServer_CPU,
RESOURSE '_tvI4Dq_EeCcBp63PfiyA@HDD' AS DBServer_HDD
USING pcm@'mediastore.properties';
```

**Results**

<table>
<thead>
<tr>
<th>Entity Type</th>
<th>Entity Alias</th>
<th>Entity Identifier</th>
<th>Metric Name</th>
<th>Res.</th>
<th>Result Value</th>
<th>Result Accu</th>
</tr>
</thead>
<tbody>
<tr>
<td>D DoF</td>
<td>Replication</td>
<td>_Q8jweEg9Ed2v5eXKb0Q9g</td>
<td>Parameter Setting</td>
<td>true</td>
<td>2.000000</td>
<td>1.0000</td>
</tr>
<tr>
<td>D DoF</td>
<td>Workload</td>
<td>'TyV-MFbW6d6ActLj8Gdl_A'</td>
<td>Parameter Setting</td>
<td>true</td>
<td>100.000000</td>
<td>1.0000</td>
</tr>
<tr>
<td>R Resource</td>
<td>AppServer_CPU</td>
<td>'5uTBUBpmEdyxpPYxt_m3w@CPU'</td>
<td>utilisation</td>
<td>true</td>
<td>0.603704</td>
<td>1.0000</td>
</tr>
<tr>
<td>R Resource</td>
<td>DBServer_CPU</td>
<td>'_tvI4Dq_EeCcBp63PfiyA@CPU'</td>
<td>utilisation</td>
<td>true</td>
<td>0.500486</td>
<td>1.0000</td>
</tr>
<tr>
<td>R Resource</td>
<td>DBServer_HDD</td>
<td>'_tvI4Dq_EeCcBp63PfiyA@HDD'</td>
<td>utilisation</td>
<td>true</td>
<td>0.714669</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Introduction of DQL

Case Study MediaStore

Fabian Gorsler, Fabian Brosig, Samuel Kounev – Controlling the Palladio Bench using DQL

Conclusion
Conclusion

✓ DQL Connector ready to use with PCM
✓ Approach to unify the interfaces to PCM, instead of:
  - EMF + PCM EA + SensorFramework
✓ Further automation with DoFs
  - Relies on PCM EA
✓ Use Case: API for PCM
  - Can be embedded or exposed as a centralized instance

Future work on goal-oriented queries
References I

Master’s thesis, Karlsruhe Institute of Technology (KIT).

Master thesis, Karlsruhe Institute of Technology (KIT).

Technical report, Karlsruhe Institute of Technology (KIT), Karlsruhe.
To interchange data, requests and results need a representation

Support of different modeling approaches requires abstraction

Encapsulation of structural model information simplifies interfaces
Backup: Mapping Meta-Model II

**Mapping**

- `modelLocation`: EString

**EntityMapping**

**DoF**

**Aggregate**

**ExtendedEntity**

- `properties`: Properties

**Entity**

- `identifier`: EString
- `alias`: EString

**Service**

**Resource**

**Probe**

- `metricName`: EString

**Result**

- `valid`: EBooleanObject

**DecimalResult**

- `value`: EBigDecimal

Generic Representation of Probes (i.e. a measurable metric) and Results

Top-Level Aggregates

Generic Representation of Model Entities

**Request**

**Response**

 identifier = "id 1"
alias = "res 1"
:Resource
identifier = "id 2"
alias = "svc 1"
:modelLocation = "void:/ /url"

**DoF**

`aggregates` 0..*

**Aggregate**

`doF` 0..*

**ExtendedEntity**

`properties` 0..*

**Entity**

**Service**

`resources` 0..*

**Resource**

**Probe**

`probes` 0..*

**Result**

**DecimalResult**

`value` 0..*

2013-11-27
DQL QEE: Implementation of External Interface

Create Instance

Execute Query

DQL QEE: Model Structure Query Interpreter

Create Instance

Interpret & Verify Query

Create Request

Execute Request

DQL Connector: Model Structure Query Connector

Process Request

Execute External Solver

Return Result Values

Create Response

DQL Connector: Model Solver Component

Compute

Return Result Values

Create Instance
Backup: Component Interaction II

DQL QEE: Implementation of External Interface

DQL QEE: Perf. Metrics Query Interpreter

DQL Connector: Perf. Metrics Query Connector

DQL Connector: Model Solver Component