Filling-the-Gap tool Manual

Imperial College London

UK

Table of Contents

1. Introduction

DevOps is a recent trend in software engineering that bridges the gap between software development and operations, putting the developer in greater control of the operational environment in which the application runs. To support Quality-of-Service (QoS) analysis, the developer may rely on software performance models. However, to provide reliable estimates, the input parameters must be continuously updated and accurately estimated. Accurate estimation is challenging because some parameters are not explicitly tracked by log files requiring deep monitoring instrumentation that poses large overheads, unacceptable in production environments.

The Filling-the-Gap (FG) tool is a component for continuous parametrization of performance models. The vision of the FG tool is described in [6]. The FG tool implements a set of statistical estimation algorithms to parameterize performance models from runtime monitoring data. Multiple algorithms are included, allowing for alternative ways to obtain estimates for different metrics, but with an emphasis on resource demand estimation. A distinguishing feature of FG tool is that it supports advanced algorithms to estimate parameters based on response times and queue-length data, which makes the tool useful in particular for applications running in virtualized environments where utilization readings are not always available. In addition, the FG tool offers support for parallel computations, integrates monitoring data acquisition, and generates performance reports.

The FG tool is consisted of 2 main components: the Local DB and the FG Analyzer. The Local DB is a local database, which is built upon the Fuseki database. The Local DB is in charge of periodically obtaining runtime monitoring data that will be used by the FG Analyzer. After receiving runtime data stored in the Local DB, the FG Analyzer provides accurate estimates to parametrise the design-time Quality-of-Service (QoS) models.

The performance model considered here are Palladio Component Model (PCM[1]) and the Layered Queueing Network (LQN[2]) models. PCM models have recently been a popular model for application performance analysis. It offers the capability of integrating application component models with resource and usage models, which can be used for QoS analysis by means of transformations to performance models, such as LQN. In the appendix we give some examples of the models.

The FG tool is under the BSD 3-Clause license.

2. Requirements

In order to install and run the Filling-the-GAP (FG) tool the requirements listed in Table 1 are needed.

Table 1 FG system requirement

Requirement	Version	Where to get it
MCR	R2013a	http://uk.mathworks.com/products/compiler/mcr/
Fuseki	1.1.1	http://jena.apache.org/documentation/serving_data/
Java	1.7.0	https://java.com/en/download/

The latest release of FG Analyzer can be found at:

[1] https://sdqweb.ipd.kit.edu/wiki/Palladio_Component_Model
[2] http://www.sce.carleton.ca/rads/lqns/lqn-documentation/

https://github.com/imperial-modaclouds/modaclouds-fg-analyzer

2.1 System requirement

FG Analyzer requires a minimum of 1G memory and 512MB disk space. It has been successfully tested with Ubuntu 12.04 and Windows 7.

3. Installation

3.1 FG local DB

Install Fuseki and Java.

3.2 FG Analyzer

Install Java and MCR.

Linux

- After installing the MCR, replace the MCR built-in old version java with the newest JRE version

```
cp -r /usr/lib/jvm/jdk1.7.0/jre/ /usr/local/MATLAB/MATLAB Compiler
Runtime/v81/sys/java/jre/glnxa64/
```

- Replace also the MCR built-in old version httpcore.jar with the latest JRE one

```
cd /usr/local/MATLAB/MATLAB Compiler Runtime/v81/java/jarext/axis2/
mv httpcore.jar httpcore backup.jar
cd <the folder containing the latest httpcore.jar>
sudo cp httpcore.jar /usr/local/MATLAB/MATLAB Compiler Runtime/v81/java/jarext/axis2/
```

Windows

- After installing the MCR, replace the MCR built-in old version java with the newest JRE version, same way as the Linux version
- Replace also the MCR built-in old version httpcore.jar with the latest JRE one, same way as the Linux version

4. Configuration

4.1 FG local DB

There is no system or application configuration for FG local DB.

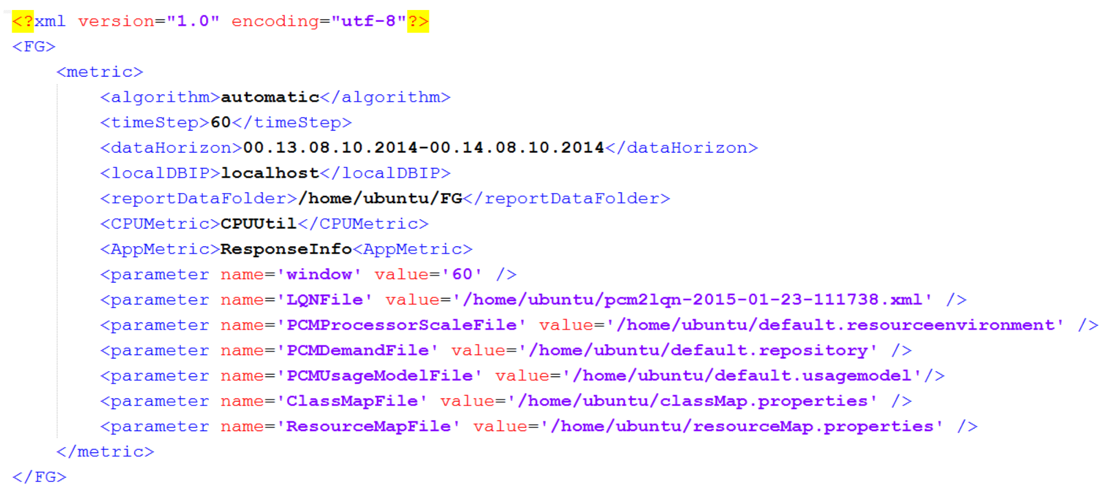
4.2 FG Analyzer

The FG Analyzer reads the configuration settings from a configuration file. The configuration file contains the following parameters:

- ***algorithm***: The algorithm to use for demand estimation. Options are:
  ci [1], *minps* [2], *erps* [2], *gql* [3], *ubr* [4], *ubo* [5] and automatic. The brief introduction to each algorithm is introduced in the Appendix.
  If put "automatic", the FG Analyzer will automatically choose the algorithm according to the monitoring data at run time. This optional can be useful when the user is not clear about the details of the algorithms and their requirement.
- ***timeStep***: The time interval to execute the analysis in seconds (e.g. 60).
- ***dataHorizon***: The horizon of data to be used. If put a single integer, such as 5, it means is will use the data generated from the last 5 minutes. You can also give a range such as 00.13.08.10.2014-00.14.08.10.2014. The format is minute.hour.day.monty.year.

- ***CPUMetric***: The metric name for CPU Utilization. Since the monitoring data saved in the local DB may have different names. User is suggested to put down here the metric name for CPU utilization.
- ***AppMetric***: The metric name for the application response time. The reason to have this metric is the same for the *CPUMetric*.
- ***localDBIP***: The IP of the local database (e.g. localhost).
- ***reportDataFolder***: The folder to keep the generated report.
- ***window***: The window parameter (in seconds) for the common data format. The common data format, which is used to standardize the estimation procesure, is described in the Appendix.
- ***LQNFile***: The path to the QoS model file.
- ***PCMProcessorScaleFile***: The path to the PCM model that determines the processor rate.
- ***PCMDemandFile***: The path to the PCM model that defines the resource demand.
- ***PCMUsageModelFile***: The path to the PCM model that defines the users.
- ***ClassMapFile***: The mapping file of job classes between LQN and PCM models.
- ***ResourceMapFile***: The mapping file of resources between LQN and PCM models. The details of the classMap file and resourceMap file are in the Appendix.

Figure 1 shows an example of the configuration file of the FG tool. It can be noticed that this is an xml file containing the parameters we have introduced above.



```xml
<?xml version="1.0" encoding="utf-8"?>
<FG>
    <metric>
        <algorithm>automatic</algorithm>
        <timeStep>60</timeStep>
        <dataHorizon>00.13.08.10.2014-00.14.08.10.2014</dataHorizon>
        <localDBIP>localhost</localDBIP>
        <reportDataFolder>/home/ubuntu/FG</reportDataFolder>
        <CPUMetric>CPUUtil</CPUMetric>
        <AppMetric>ResponseInfo<AppMetric>
        <parameter name='window' value='60' />
        <parameter name='LQNFile' value='/home/ubuntu/pcm2lqn-2015-01-23-111738.xml' />
        <parameter name='PCMProcessorScaleFile' value='/home/ubuntu/default.resourceenvironment' />
        <parameter name='PCMDemandFile' value='/home/ubuntu/default.repository' />
        <parameter name='PCMUsageModelFile' value='/home/ubuntu/default.usagemodel'/>
        <parameter name='ClassMapFile' value='/home/ubuntu/classMap.properties' />
        <parameter name='ResourceMapFile' value='/home/ubuntu/resourceMap.properties' />
    </metric>
</FG>
```

Figure 1 Example of configuration file

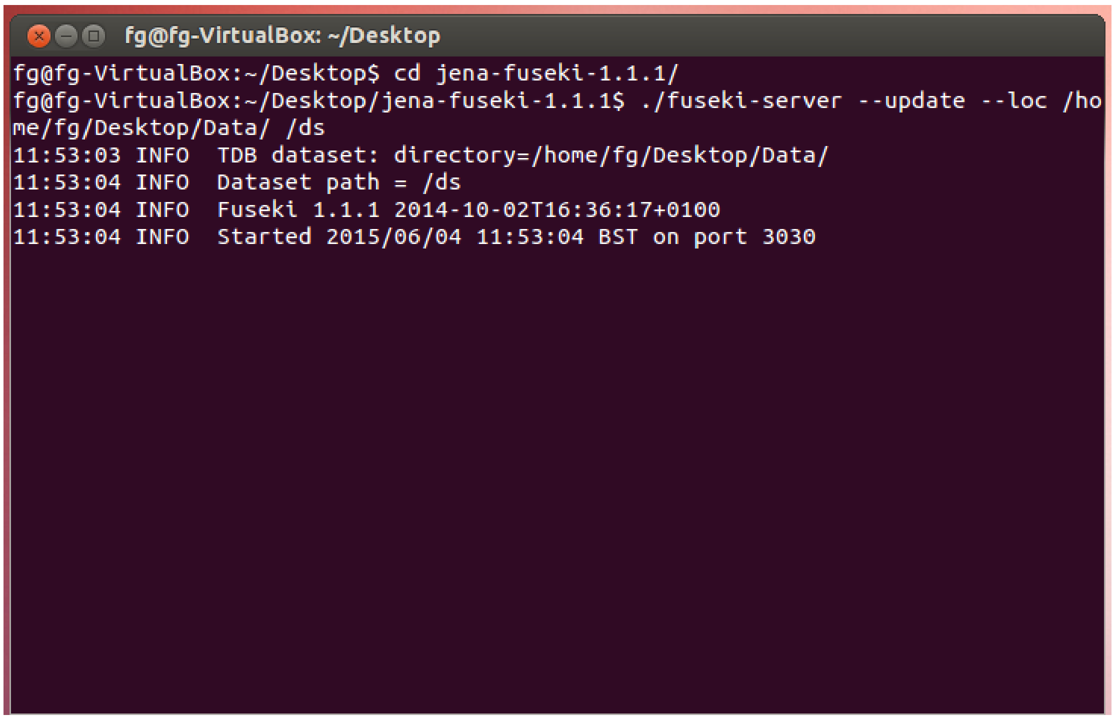
5. Usage

5.1 FG local DB

- Start the fuseki database:
  Linux
  `./fuseki-server --update --loc <DataFolder> /ds`
  Windows
  `Java –jar fuseki-server.jar --update --loc <DataFolder> /ds`

  where
  <DataFolder>: the fuseki database storage folder, for instance "/tmp/data".

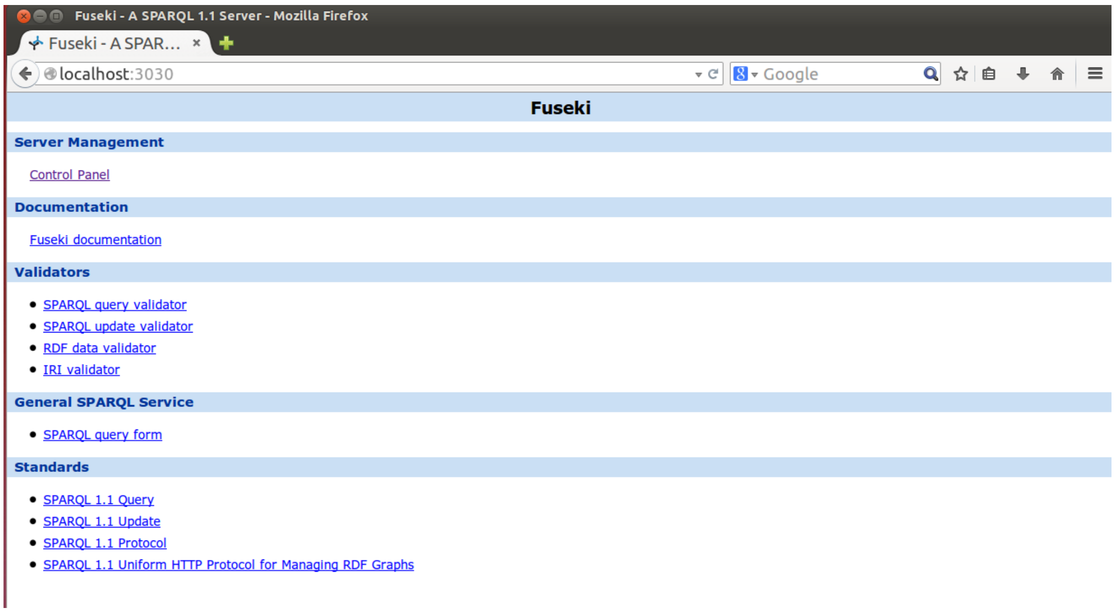
Figure 2 is a screen shot of starting the fuseki database on Ubuntu 12.04. We choose the "/home/fg/Desktop/Data" as the database storage folder.



```
fg@fg-VirtualBox: ~/Desktop
fg@fg-VirtualBox:~/Desktop$ cd jena-fuseki-1.1.1/
fg@fg-VirtualBox:~/Desktop/jena-fuseki-1.1.1$ ./fuseki-server --update --loc /ho
me/fg/Desktop/Data/ /ds
11:53:03 INFO  TDB dataset: directory=/home/fg/Desktop/Data/
11:53:04 INFO  Dataset path = /ds
11:53:04 INFO  Fuseki 1.1.1 2014-10-02T16:36:17+0100
11:53:04 INFO  Started 2015/06/04 11:53:04 BST on port 3030
```

Figure 2 Example of starting fuseki

Once fuseki is started, you can visit http://locahost:3030 to access the user interface. Figure 3 shows a screen shot of the user interface.



Fuseki - A SPARQL 1.1 Server - Mozilla Firefox

localhost:3030

Fuseki

Server Management

Control Panel

Documentation

Fuseki documentation

Validators

- SPARQL query validator
- SPARQL update validator
- RDF data validator
- IRI validator

General SPARQL Service

- SPARQL query form

Standards

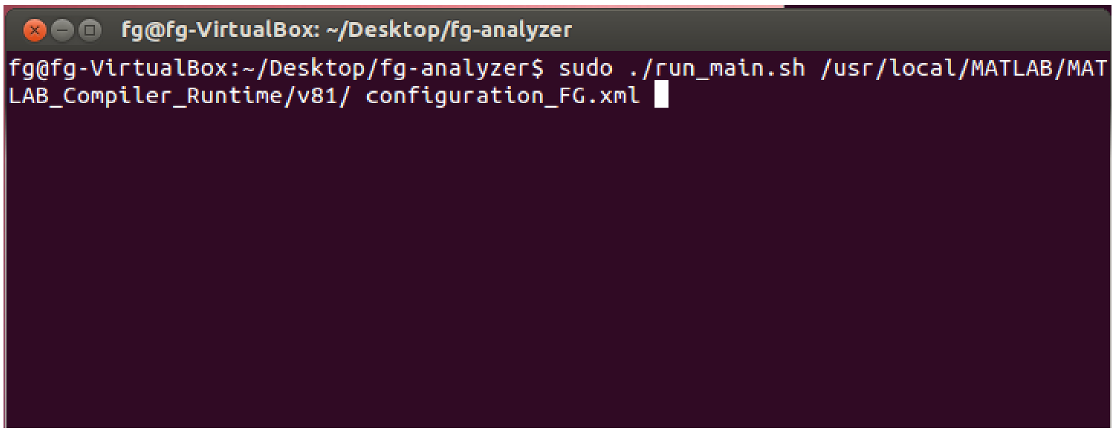
- SPARQL 1.1 Query
- SPARQL 1.1 Update
- SPARQL 1.1 Protocol
- SPARQL 1.1 Uniform HTTP Protocol for Managing RDF Graphs

Figure 3 Fueski user interface

5.2 FG Analyzer

- Start the main of the FG Analyzer
  Linux
  `./run_main <MCRPath> <file>`
  Windows
  `Main.exe <file>`

  where
  <MCRPath>: the path to the Matlab Compiler Runtime environment.
  <file>: the configuration file
  Figure 4 shows a screen shot of starting the FG Analyzer on Ubuntu 12.04.



```
fg@fg-VirtualBox: ~/Desktop/fg-analyzer
fg@fg-VirtualBox:~/Desktop/fg-analyzer$ sudo ./run_main.sh /usr/local/MATLAB/MAT
LAB_Compiler_Runtime/v81/ configuration_FG.xml
```

Figure 4 Example of starting FG Analyzer

6. How to populate the database

- Start to populate the database :
  `Java –jar HDBDataGenerator-0.0.1-SNAPSHOT.jar <fileName> <period>`

  where
  <fileName>: the name of the file containing the trace
  <period>: the period in milliseconds to save data into the database

The file should follow the pattern:

```
resourceName   metric   metricValue     timestamps
```

where:
resourceName: the resource where the monitoring data is collected. This is used to differentiate different resources. For instance, CPU Utilization can be both collected at VM1 and VM2. To differentiate them, the resource has to be named, e.g. VM1 or VM2.
metric: the metric name, such as CPUUtilization.
metricValue: the value of the corresponding metric.
timestamps: the timestamps when the particular *metricValue* is collected.

For the response time data, the *metricValue* should follow the pattern:

```
requestClass,arrivalTimestamps,responseTime
```

where:
requestClass: the request name, such as “Login”.
arrivalTimestamps: the arrival time of the corresponding request.

responseTime: the response time of the corresponding request.

Figure 5 shows a screen shot of a sample file of the data.

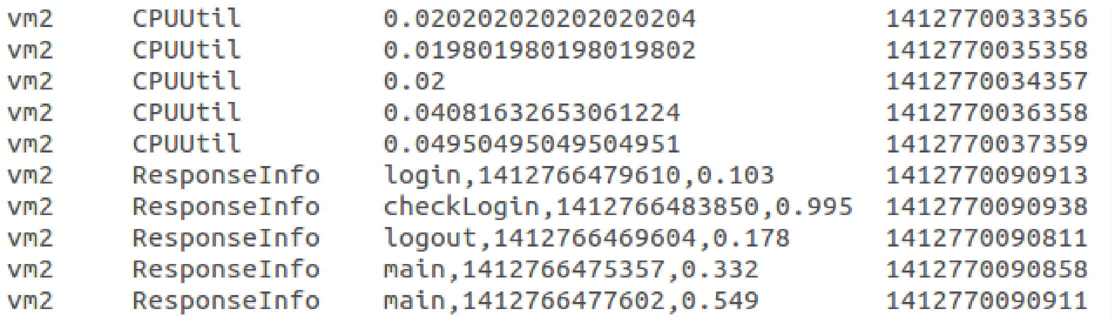

```
vm2     CPUUtil         0.020202020202020204            1412770033356
vm2     CPUUtil         0.019801980198019802            1412770035358
vm2     CPUUtil         0.02                            1412770034357
vm2     CPUUtil         0.04081632653061224             1412770036358
vm2     CPUUtil         0.04950495049504951             1412770037359
vm2     ResponseInfo    login,1412766479610,0.103       1412770090913
vm2     ResponseInfo    checkLogin,1412766483850,0.995  1412770090938
vm2     ResponseInfo    logout,1412766469604,0.178      1412770090811
vm2     ResponseInfo    main,1412766475357,0.332        1412770090858
vm2     ResponseInfo    main,1412766477602,0.549        1412770090911
```

Figure 5 Example of sample file

References

[1] Pérez, J.F., Pacheco-Sanchez, S. and Casale, G. An Offline Demand Estimation Method for Multi-Threaded Applications. Proceedings of MASCOTS 2013, 2013

[2] Perez, J.F., Casale, G, and Pacheco-Sanchez, S. "Estimating Computational Requirements in Multi-Threaded Applications." IEEE Transaction on Software Engineering. 2014.

[3] Wang, W., and Casale, G. Bayesian service demand estimation with Gibbs sampling." Proceedings. of IEEE MASCOTS. 2013.

[4] Zhang, Q., Cherkasova L., and Smirni, E. A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. Proceedings of the Fourth International Conference on Autonomic Computing, 2007.

[5] Liu, Z., Wynter, L., Xia, C. H. and Zhang, F. Parameter inference of queueing models for IT systems using end-to-end measurements. Performance Evaluation, Elsevier, 2006.

[6] Perez, J.F., Wang, W., Casale, G. Towards a Devops approach for software quality engineering. WOSP, 2015.

Appendices

A. Common data format

To standardise the use of the estimation algorithms, we have adopted a common data format, from which each algorithm can select the data it requires to perform the estimation. We assume the data has been or is being collected for an application that provides a number of different services, grouped in service classes. There are a total of K different service classes, for which data is collected either in time windows, or for each request, or both. The data format is a data structure with 6 rows and K + 1 columns. Each of the first K columns corresponds to a service class, while the last columns is reserved for overall data, disregarding the service class.

For each column k, the information in each row is the following:

1. Sampling interval (in milliseconds): holds the timestamps corresponding to the end of each sampling interval. Assuming the data corresponds to N sampling intervals, column k in this first row must have an array of size N.

2. Utilization (normalized between 0 and 1): holds the average CPU utilization for each sampling window. Typically only overall CPU utilization is collected, thus only the column K +1 will hold an array of size N, while the other columns will be empty.
3. Arrival times (in milliseconds): holds the timestamps of the arrival of each class-k request. In this case, each column holds an array that can be of different size, as a different number of requests may have arrived during the whole observation period.
4. Response times (in seconds): holds the observed response times (departure time minus arrival time) of each request. The size of the arrays in each column can differ from each other, just as with the arrival times.
5. Average response time (in seconds): holds the mean response time of the requests processed in each sampling window. In this case each column holds an array of size N. If no requests of a given class are processed in a sampling interval, the corresponding entry in the array is set to zero.
6. Throughput (in 1/second): holds the throughput observed for each service class in each sampling window. The throughput is computed as the total number of requests processed in the sampling interval, divided by the length of the interval (in seconds). Each column in this row holds an array of size N.

Notice that rows 1, 2, 5, and 6, keep measures for each sampling window, so that each entry in these columns holds an array of size N, where N is the number of sampling windows. Instead, rows 3 and 4 keep measurements for specific requests, and thus the entries in these rows holds arrays of variable size.

B. Estimation algorithms

- CI [1]: the Complete Information (CI) method requires a full trace of the requests, that is, the times at which every request arrives and departs from the resource.
- MINPS [2]: the MINPS method is a maximum likelihood method based on a Markov Chain representation of the response time given the observed queue length. It requires response times and queue lengths observed upon arrival.
- ERPS [2]: the Extended Regression-Based (RPS) approach makes use of the response times and queue lengths observed at arrival times.
- GQL [3]: the Gibbs sampling based on Queue Lengths (GQL) method uses queue-length samples collected at run time to estimate the resource demand by using the Bayes' theorem.
- UBR [4]: the Utilization-Based Regression method takes input of the CPU utilization and the throughput to estimate the resource demand with linear regression.
- UBO [5]: the Utilization-Based Optimization method makes use of the CPU utilization, throughput as well as the average response time to estimate the resource demand.

The required input of the algorithms are listed in Table 2.

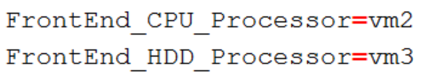
Table 2 Required data for different estimation algorithms.

Output	Data Required	Algorithm
Resource demand	Full trace	CI
	Response Times Queue length (arrival)	MINPS ERPS
	Queue length	GQL

	Utilization Throughput	UBR UBO[3]

C. classMap and resourceMap files

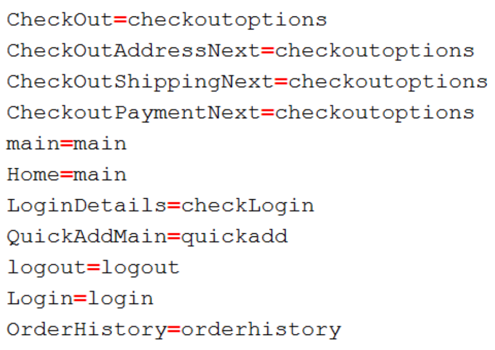
Resource name is defined in the deployment model and typically stands for the virtual machine name. There could be inconsistent names for the ones defined in the models and the ones from the monitoring data. Figure 6 shows an example of the resourceMap file.



```
FrontEnd_CPU_Processor=vm2
FrontEnd_HDD_Processor=vm3
```

Figure 6 Examples of resourceMap file

Class means the request names for the application. It can be different for the name defined in the deployment model and the actually application. For the ClassMapFile, it is a properties file, of which the left column is the request defined in the performance models and the right column is from the application. Figure 7 shows an example of the classMap file.



```
CheckOut=checkoutoptions
CheckOutAddressNext=checkoutoptions
CheckOutShippingNext=checkoutoptions
CheckoutPaymentNext=checkoutoptions
main=main
Home=main
LoginDetails=checkLogin
QuickAddMain=quickadd
logout=logout
Login=login
OrderHistory=orderhistory
```

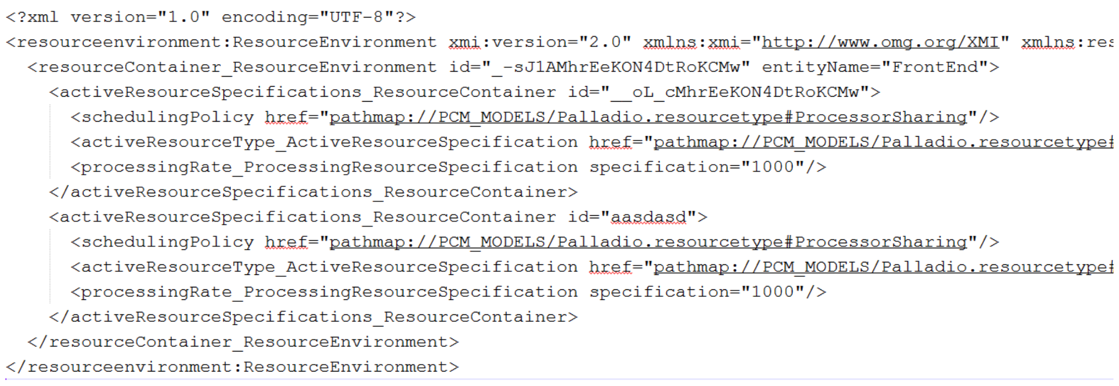
Figure 7 Examples of the classMap file

D. PCM and LQN models

Here we give some examples of PCM and LQN models.

Below is an example of PCM resource environment file congaing the processing speed of each resource. This is used to scale the resource demand. Figure 8 shows a screenshot of an example of the PCM resource environment file. The processing rate of the server is inside the field "processingRate_ProcessingResourceSpecification".

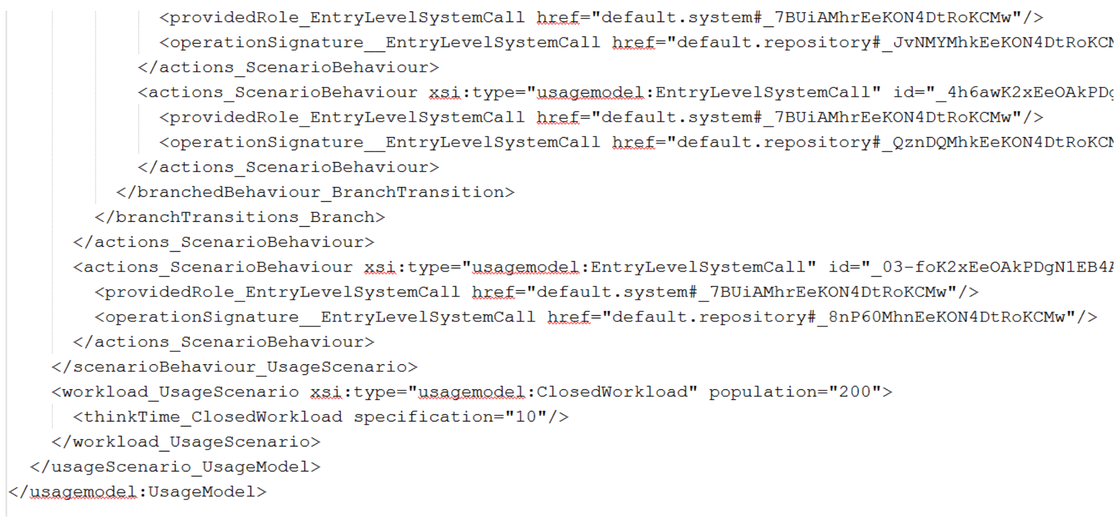
[3] This method also requires average response times



```
<?xml version="1.0" encoding="UTF-8"?>
<resourceenvironment:ResourceEnvironment xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:res
  <resourceContainer_ResourceEnvironment id="_-sJ1AMhrEeKON4DtRoKCMw" entityName="FrontEnd">
    <activeResourceSpecifications_ResourceContainer id="__oL_cMhrEeKON4DtRoKCMw">
      <schedulingPolicy href="pathmap://PCM_MODELS/Palladio.resourcetype#ProcessorSharing"/>
      <activeResourceType_ActiveResourceSpecification href="pathmap://PCM_MODELS/Palladio.resourcetype#
      <processingRate_ProcessingResourceSpecification specification="1000"/>
    </activeResourceSpecifications_ResourceContainer>
    <activeResourceSpecifications_ResourceContainer id="aasdasd">
      <schedulingPolicy href="pathmap://PCM_MODELS/Palladio.resourcetype#ProcessorSharing"/>
      <activeResourceType_ActiveResourceSpecification href="pathmap://PCM_MODELS/Palladio.resourcetype#
      <processingRate_ProcessingResourceSpecification specification="1000"/>
    </activeResourceSpecifications_ResourceContainer>
  </resourceContainer_ResourceEnvironment>
</resourceenvironment:ResourceEnvironment>
```

Figure 8 Example of PCM resource environment file

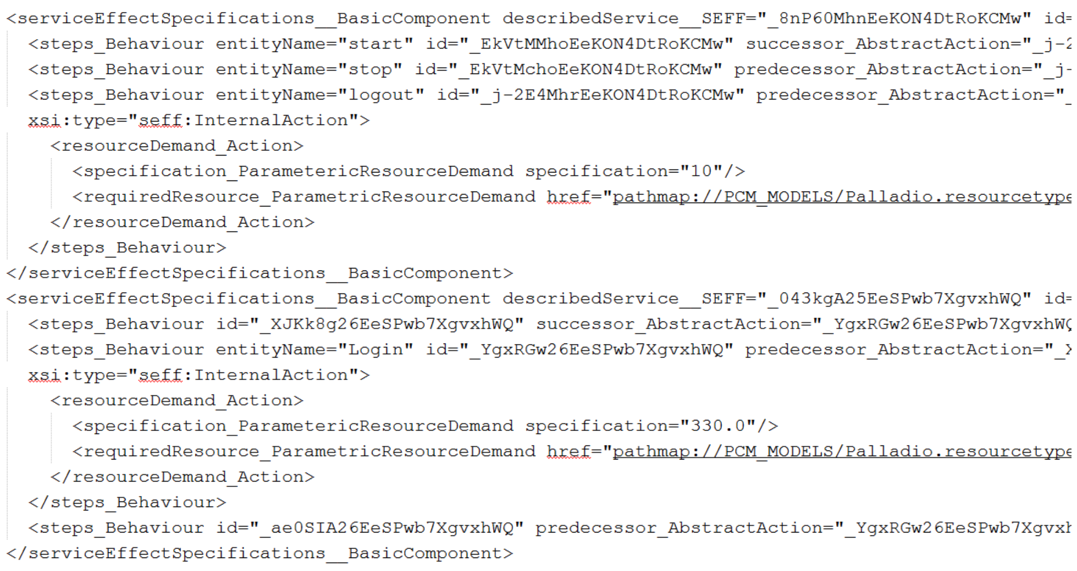
Below is part of the PCM usage model, which contains the total population and the think time of the requests. Figure 9 shows a screenshot of an example of the PCM usage model file. The population and thinktime is under the field “population” and “thinkTime_ClosedWorkload”.



```
            <providedRole_EntryLevelSystemCall href="default.system#_7BUiAMhrEeKON4DtRoKCMw"/>
            <operationSignature__EntryLevelSystemCall href="default.repository#_JvNMYMhkEeKON4DtRoKCM
          </actions_ScenarioBehaviour>
          <actions_ScenarioBehaviour xsi:type="usagemodel:EntryLevelSystemCall" id="_4h6awK2xEeOAkPDg
            <providedRole_EntryLevelSystemCall href="default.system#_7BUiAMhrEeKON4DtRoKCMw"/>
            <operationSignature__EntryLevelSystemCall href="default.repository#_QznDQMhkEeKON4DtRoKCM
          </actions_ScenarioBehaviour>
        </branchedBehaviour_BranchTransition>
      </branchTransitions_Branch>
    </actions_ScenarioBehaviour>
    <actions_ScenarioBehaviour xsi:type="usagemodel:EntryLevelSystemCall" id="_03-foK2xEeOAkPDgN1EB4A
      <providedRole_EntryLevelSystemCall href="default.system#_7BUiAMhrEeKON4DtRoKCMw"/>
      <operationSignature__EntryLevelSystemCall href="default.repository#_8nP60MhnEeKON4DtRoKCMw"/>
    </actions_ScenarioBehaviour>
  </scenarioBehaviour_UsageScenario>
  <workload_UsageScenario xsi:type="usagemodel:ClosedWorkload" population="200">
    <thinkTime_ClosedWorkload specification="10"/>
  </workload_UsageScenario>
 </usageScenario_UsageModel>
</usagemodel:UsageModel>
```

Figure 9 Example of PCM usage model

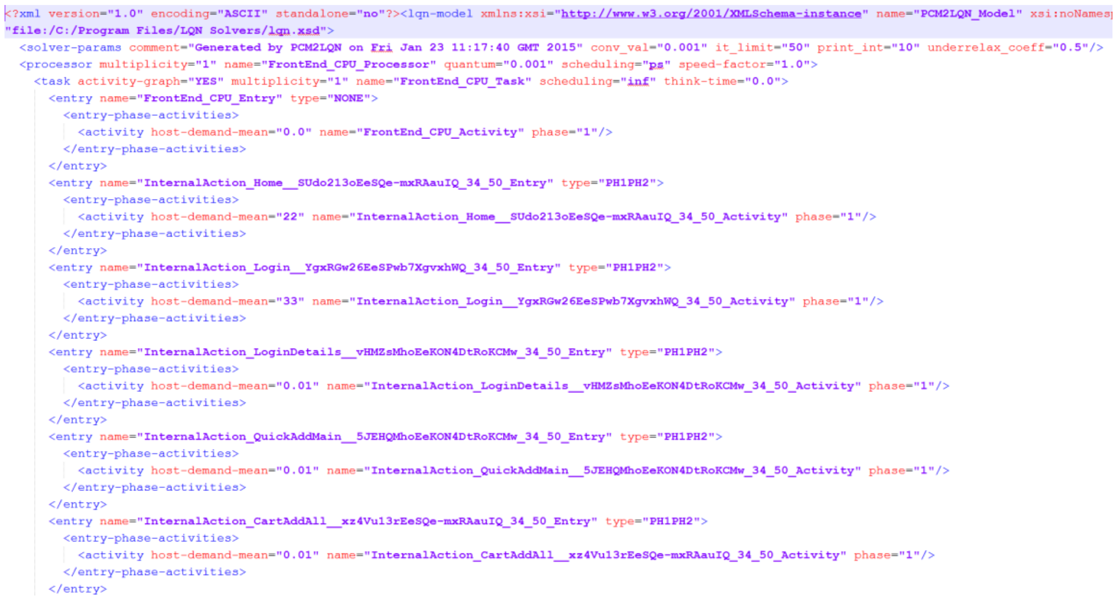
Below is the PCM repository model, which contains the service demand of the requests. Figure 10 shows a screenshot of an example of the PCM repository model file. The resource demand is shown under the field “specification_ParametericResourceDemand”.



```
<serviceEffectSpecifications__BasicComponent describedService__SEFF="_8nP60MhnEeKON4DtRoKCMw" id=
  <steps_Behaviour entityName="start" id="_EkVtMMhoEeKON4DtRoKCMw" successor_AbstractAction="_j-2
  <steps_Behaviour entityName="stop" id="_EkVtMchoEeKON4DtRoKCMw" predecessor_AbstractAction="_j-
  <steps_Behaviour entityName="logout" id="_j-2E4MhrEeKON4DtRoKCMw" predecessor_AbstractAction="_
  xsi:type="seff:InternalAction">
    <resourceDemand_Action>
      <specification_ParametericResourceDemand specification="10"/>
      <requiredResource_ParametricResourceDemand href="pathmap://PCM_MODELS/Palladio.resourcetype
    </resourceDemand_Action>
  </steps_Behaviour>
</serviceEffectSpecifications__BasicComponent>
<serviceEffectSpecifications__BasicComponent describedService__SEFF="_043kgA25EeSPwb7XgvxhWQ" id=
  <steps_Behaviour id="_XJKk8g26EeSPwb7XgvxhWQ" successor_AbstractAction="_YgxRGw26EeSPwb7XgvxhWQ
  <steps_Behaviour entityName="Login" id="_YgxRGw26EeSPwb7XgvxhWQ" predecessor_AbstractAction="_X
  xsi:type="seff:InternalAction">
    <resourceDemand_Action>
      <specification_ParametericResourceDemand specification="330.0"/>
      <requiredResource_ParametricResourceDemand href="pathmap://PCM_MODELS/Palladio.resourcetype
    </resourceDemand_Action>
  </steps_Behaviour>
  <steps_Behaviour id="_ae0SIA26EeSPwb7XgvxhWQ" predecessor_AbstractAction="_YgxRGw26EeSPwb7Xgvxh
</serviceEffectSpecifications__BasicComponent>
```

Figure 10 Example of PCM repository model

Below is the corresponding LQN model for the above PCM model. It can be noticed the service demand is defined for each type of request. Figure 10 shows the screenshot of an example of the LQN file. The resource demand is under the field "host-demand-mean", which represents the average resource demand of that request.



```
<?xml version="1.0" encoding="ASCII" standalone="no"?><lqn-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="PCM2LQN_Model" xsi:noNames
"file:/C:/Program Files/LQN Solvers/lqn.xsd">
  <solver-params comment="Generated by PCM2LQN on Fri Jan 23 11:17:40 GMT 2015" conv_val="0.001" it_limit="50" print_int="10" underrelax_coeff="0.5"/>
  <processor multiplicity="1" name="FrontEnd_CPU_Processor" quantum="0.001" scheduling="ps" speed-factor="1.0">
    <task activity-graph="YES" multiplicity="1" name="FrontEnd_CPU_Task" scheduling="inf" think-time="0.0">
      <entry name="FrontEnd_CPU_Entry" type="NONE">
        <entry-phase-activities>
          <activity host-demand-mean="0.0" name="FrontEnd_CPU_Activity" phase="1"/>
        </entry-phase-activities>
      </entry>
      <entry name="InternalAction_Home__SUdo213oEeSQe-mxRAauIQ_34_50_Entry" type="PH1PH2">
        <entry-phase-activities>
          <activity host-demand-mean="22" name="InternalAction_Home__SUdo213oEeSQe-mxRAauIQ_34_50_Activity" phase="1"/>
        </entry-phase-activities>
      </entry>
      <entry name="InternalAction_Login__YgxRGw26EeSPwb7XgvxhWQ_34_50_Entry" type="PH1PH2">
        <entry-phase-activities>
          <activity host-demand-mean="33" name="InternalAction_Login__YgxRGw26EeSPwb7XgvxhWQ_34_50_Activity" phase="1"/>
        </entry-phase-activities>
      </entry>
      <entry name="InternalAction_LoginDetails__vHMZsMhoEeKON4DtRoKCMw_34_50_Entry" type="PH1PH2">
        <entry-phase-activities>
          <activity host-demand-mean="0.01" name="InternalAction_LoginDetails__vHMZsMhoEeKON4DtRoKCMw_34_50_Activity" phase="1"/>
        </entry-phase-activities>
      </entry>
      <entry name="InternalAction_QuickAddMain__5JEHQMhoEeKON4DtRoKCMw_34_50_Entry" type="PH1PH2">
        <entry-phase-activities>
          <activity host-demand-mean="0.01" name="InternalAction_QuickAddMain__5JEHQMhoEeKON4DtRoKCMw_34_50_Activity" phase="1"/>
        </entry-phase-activities>
      </entry>
      <entry name="InternalAction_CartAddAll__xz4Vu13rEeSQe-mxRAauIQ_34_50_Entry" type="PH1PH2">
        <entry-phase-activities>
          <activity host-demand-mean="0.01" name="InternalAction_CartAddAll__xz4Vu13rEeSQe-mxRAauIQ_34_50_Activity" phase="1"/>
        </entry-phase-activities>
      </entry>
```

Figure 11 Example of LQN model