

Instructions and Usage of PAVO

Master Traineeship of

Maximilian König

At the Department of Computer Science
Chair for Computer Science II
Software Engineering



Advisor: Dipl. Inf. Jürgen Walter

Duration: 1. September 2016 – 28. February 2017

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 2 | Approach | 3 |
| 2.1 | Goals | 3 |
| 2.2 | Interface to Descartes Query Language | 5 |
| 3 | Implementation | 7 |
| 3.1 | Graphical User Interface | 7 |
| 3.2 | Chart Types | 7 |
| 3.2.1 | Confidence Intervals | 7 |
| 3.2.2 | Bar Charts | 8 |
| 3.2.3 | Line Charts | 8 |
| 3.2.4 | Difference Chart | 8 |
| 3.2.5 | Scatter Plot | 8 |
| 3.2.6 | Box Plot | 8 |
| 3.2.7 | Bubble Chart | 8 |
| 3.2.8 | Heat Map | 9 |
| 3.2.9 | Histogram | 9 |
| 3.2.10 | Pie Chart | 9 |
| 3.3 | Result Type | 9 |
| 3.3.1 | Abstract Quantitative Result | 9 |
| 3.3.2 | Result Container | 9 |
| 3.3.3 | Series Result | 9 |
| 3.3.4 | String Result | 11 |
| 3.3.5 | Value Result | 11 |
| 3.4 | Diagram Visualization | 11 |
| 3.4.1 | Graphics Engine | 11 |
| 3.4.2 | Diagram Visualizer | 11 |
| 3.5 | Diagram Export | 11 |
| 4 | Summary | 11 |
| | Bibliography | 13 |
| 5 | Acronyms | 13 |

1. Introduction

Performance is of particular relevance to software system design, operation, and evolution because it has a major impact on key business indicators. That is why it is interesting to create tools to simulate computer systems and check for example their workload with different applications. Beside simulation one needs to compare such systems or models to each other. The visualization of results is very important to recognize problems. Common tools for performance predictions require a large amount of experience with modeling formalism.

The idea of Declarative Performance Engineering (DPE) defines that using a system shall be as easy as possible. The goal of DPE is that the user just has to ask what he wants and the system automates the how and returns the result.

In applications, just as Queueing Petri net Modeling Environment (QPME), which analyze computing systems, is a visualization for the result one get from his request existing. These result representations are only fitted for the appropriate software package. But all these visualizations do the same thing. So why not implement a generic library called Performance Analyse VisualizatiOn (PAVO) library, which supports a visualization, that can be used for multiple tools. At the chair of software engineering at the University in Würzburg a language to direct queries to models was developed. That language is Descartes Query Language (DQL). This approach to performance predictions shall automate the process of extracting informations about these computer systems. It should minimize the manual work done by the analyst and support him with software. The problem with the representation of results is that these are difficult to understand, cause they are shown in complex log files. DQL at least filters them and puts the most important ones, because the user asked for, in a table. Although the result visualization in a table is better than the one in files, it is not optimal. For human beings a graphical representation is much easier to understand. Also a diagram may deliver more information than a simple table. Even a measurement based analysis or a model based ones need the same types of visualization. To support such a visualization the necessary independent library was implemented in 2016. According to such a visualization software many questions need to be answered. For instance which diagram types shall be supported, how different diagrams can be compared the easiest way or should one be able to switch between different types. So we developed a tool named PAVO, an independent and easy to use library for the graphic representation for the results of DQL. That work was based on the idea of DPE to make programs as easy as possible. We use the language DQL for the evaluation and integrate the developed library to the current engine. That visualization tool is easy to extend and to integrate into different tools.

2. Approach

In this chapter we take a closer look to the goals and the approach for PAVO, how to reach them. In Section 2.1 one can see the major features which PAVO provides. After that we show how easy PAVO can be included into an existing software package. Here we take DQL as an example.

2.1 Goals

The main goal was to develop a software to visualize the results of a performance analysis program, for example DQL, according to the query with an automatically chosen diagram type. The interface will help the user to easier work with his models in the way of Declarative Performance Engineering. We also added the support of live diagrams for time changing values. The final software shall be used as a library which can be included easily into every other tool, if the return and data transport types as described in Section 2.2 are available.

It shall be possible for the user to visualize scenarios online and offline. In addition merging of different diagrams and swapping the diagram type should also be supported. By supporting this type of merge it will be much easier to compare different queries and extracting some information out of a diagram, than just using numbers in a table. The Subgoals to reach while this thesis are mentioned and explained in the following:

- **Goal 1: Automated Diagram Type Choice**

One main feature of the software is the automated choice of the matching diagram according to the query asked. For example if one got a sample of about 10 values, it is much better to see them in a line chart than using a box plot. In contrast, by receiving more than 100 values we better extract the information out of a box plot than out of a line chart with too many dots in it.

- **Goal 2: Support of Manual Diagram Switch**

In addition the user shall be able to display the values in a diagram type if he wants. This can be useful if he wants to see the received information in another way.

- **Goal 3: Support Diagram Merging**

Another feature is to merge different kinds of charts into one single diagram. This helps to compare different analysis results.

- **Goal 4: Support Offline and Online Scenarios**

One special feature our software shall offer are on one hand offline scenarios which show one single moment in time. On the other hand online plotting is possible as well to visualize an interval of time.

- **Goal 5: Support for Confidence Intervals**

Some queries asked in DQL return confidence interval for the result values. The diagrams created by the visualization of this thesis shall support such intervals and show them in the charts as error bars.

- **Goal 6: Export Feature**

The software shall offer the function to export the shown diagram as an image file and save it to the current file system. Also the size of this image shall be possible to choose by the user.

- **Goal 7: Non-functional Aims**

The non-functional aims to reach are integrating the software, which creates the diagrams in the existing tool chain of DQL at the Chair of software engineering in Würzburg. According to this integration the code and the software itself has to be kept as extensible as possible, so maybe a live-system or even the support for new diagrams because of new queries, that can be done, can be added. Another point for the extendibility is that in future work the result visualization can be integrated into a complete graphical user interface with the model and the appropriate request and the diagram and maybe some for information as output.

Now we will take a closer look to the single Subgoals. According to the Automated Diagram Choice (Goal 1) it must be said that for the current state of the thesis there are three major types of diagrams to support: line charts, box plots and bar charts. Additionally PAVO shall be able to display decimal results and string results as text and integrate the decimal ones into line charts.

By taking a closer look at DQL and the queries that can be made, it is easy to see that there are multiple scenarios we need to cover with diagrams. In the following it is explained which kind of diagram will be used to show the result of a certain query.

The first type of queries that can be asked for are those one getting samples of different values for a single moment in the network. For example a query for workload of different entities. These can be shown easily in a bar diagram, so it is possible to compare these entities. Box plot will not be chosen as default type, because it is a very special type of diagram. The standard types that are shown in the beginning are the line chart for continuous results and the bar chart for discrete ones. Secondly, the most common queries are time based (Goal 4). Here the user wants to know how a parameter of a single entity changes over time. For example the workload of entity X for some time. To show this best, the implemented Graphical User Interface (GUI) will use simple line charts. The time will be shown at the horizontal axis and the parameter asked for, e.g. the workload is shown vertically. This offers the user to best compare the parameter values over time. The last type we want to include in the GUI is optional for this thesis and can also be seen as future work to do. To see the performance of a network and its changes over time immediately, the feature of a diagram that is updating itself live, while the network model is created. If the user switches a parameter in the model he shall see the changes instantly in the diagram. So the live-system will just be an extension for the secondly named time-based queries result visualization. The diagram switches (Goal 2) shall be possible with the usage of buttons and for example combo boxes. Here the user can decide whether he wants the standard line or bar chart, a box plot, a line chart with confidence intervals as error bars or even a bar chart with error bars.

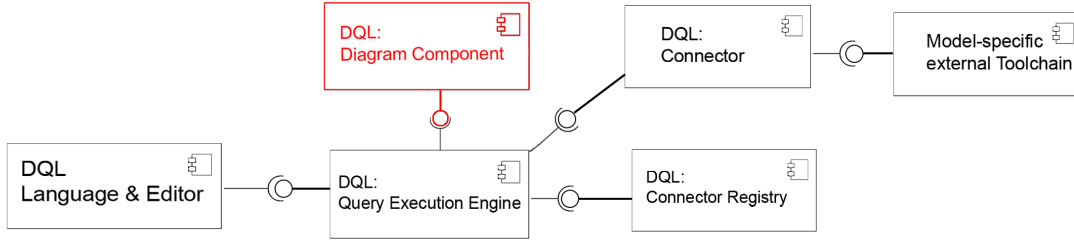


Figure 2.1: Inclusion of the Diagram Component into the Existing Toolchain of DQL

The diagram merging feature (Goal 3) should work as follows. The first result received decides which kind of diagram is shown. After that the rest of the results are merged and transformed into this diagram. For each resource received a single diagram will be created. The functionality of exporting the charts will be focused on the export of portable network graphics (png) files. If the license of the chosen chart drawing library allows to export vector graphics this would be the optimal case.

2.2 Interface to Descartes Query Language

To communicate with DQL requests our program needs to receive the data from them. There are multiple options to do that, for example JSON, XML or just by an object created. For example we need to redirect the information about the kind of result (to choose the right diagram type) or the values for the diagrams. There are two options to regard intently. The first one is XML, as used by QPME. Here all necessary information can be described in the file which is read by the chart creation program. Also the title of the diagram shall be given by an XML file. The second alternative is the passing of objects through the toolchain. In the current DQL results of a query are saved as objects of the class EntityMapping. Just because the current software provides this class it is the most easiest way to transport our data. The library will be implemented that way, that one needs to transfer his own data into the PAVO result type, to make use of it. For DQL we transformed the EntityMapping object. After taking a look at the difficulty of the implementation of the different data transport possibilities, we decided to choose the object creating path in cooperation with the extension of EntityMapping. All in all the created PAVO library can be used for every kind of program later on, which returns objects of this class. If for future work any more information is necessary, it is also simple to extend EntityMapping.

According to the point, where the diagram creation component shall be included into the existing toolchain of DQL to evaluate the library, it must be said, that it is most common, to connect it directly to the "DQL: Query Execution Engine" (Figure 4), which offers the result as EntityMapping. Another reason for this connection point is that we want to create PAVO, that works for every DQL version, for every connector and also with extern software. The opportunity to include the program as a connector is not appropriate to the named goal.

In addition the software shall not be developed as a component which can only be used with DQL. The visualization will be coded in a separate package in the repository. It is called tools.descartes.PAVO. So the visualization shall be able to be used with any other software, if the programmer transformed his result type as we do with the EntityMapping of DQL.

3. Implementation

In the following we present the implementation of the PAVO library, especially the used software pattern. According to that, we decided on a Model-View Controller pattern in combination with a Decorator pattern for the changes of the graphical user interface, depending on the current diagram type. For the export feature, we decided not to include it right into the PAVO package, because of different licenses. Here an extension point was implemented. This one is used by four different export controllers at the moment. Even more export types can be added in the future.

3.1 Graphical User Interface

The GUI was splitted into the main overlay and the export frame. The export frame is kept as simple as possible, with a combobox to choose your type of export and if nessecary the dimensions of the created image. Furthermore there is the finish button to complete the export procedure and a cancel button to stop it.

The main GUI is structured as follows: In the top there is a tab bar, separating the diagrams via metrics. Beneath that there is the String of the current query shown. Moving downwards one can find the diagram including its legend and at the bottom of the frame there are the button controls for the diagram. These controls include a button and a combo box for the diagram switch, one to export a chart and even more buttons, combo boxes and radio buttons depending on the actual chart.

3.2 Chart Types

In this section all different diagrams types, supported by PAVO are explained in detail. On the one hand side the result types supported and on the other hand side the controls shown by each kind of chart. The standard type, which can be used every time is the bar chart. In figure 3.1 one can see the decision tree of PAVO. It shows which diagram is supported in which case.

3.2.1 Confidence Intervals

One additional feature we added to PAVO are the confidence intervals that can be shown. Therefore we added the confidence intervals to the EntityMapping of the result type of

DML. Also for the types of bar chart and line chart there is an active additional radio button, which causes the diagram to switch to the same type but adds these intervals to the plot.

3.2.2 Bar Charts

As already mentioned the main type of diagram is bar chart, which supports all different result types possible. The series are shown in multiple colors. As a special feature for bar charts the user can enable the minimum, maximum and mean value of each series to be shown in the plot as horizontal lines.

3.2.3 Line Charts

Another standard type of chart is the simple line chart. It also supports the confidence intervals and the radio buttons for minimum, maximum and mean. So there is no big difference compared to a bar chart, except the fact, that a line chart can only be used, if the delivered data series are all marked as continuous and not a single one for the current metric as discrete.

3.2.4 Difference Chart

To extend the line chart type we added Difference Chart to PAVO. This kind of diagram shall show the user the difference between two series. It is enabled, if the metric has exactly two series. It is not necessary that both of them are continuous. In the plot the two series are shown as line charts and the space between the two lines is marked in another color. Here no additional features are implemented, neither the minimum, maximum and mean, nor the confidence intervals.

3.2.5 Scatter Plot

The next standard plot is the Scatter Plot, which is also supported by all different kinds of results. The points in the plot are shown as different geometric forms, depending to the series they belong to, they belong to. Scatter plot can be used to see the spreading around a single point, e.g. the center of both chart axis. As Difference Chart does not support any more features than displaying the values, so does Scatter Plot.

3.2.6 Box Plot

The last simple type of diagram supported by PAVO is the box plot. Here the program takes all y values of each series together and puts them into a box plot. Box plot is only enabled when there is more than one series of the actual metric, otherwise there would be no additional information gained by using this kind of diagram. As the last few diagram types do not support additional features, box plot also has no extras.

3.2.7 Bubble Chart

One more complex type of diagram supported by PAVO is Bubble Chart. It is a three dimensional representation of values. It was designed for showing different degrees of freedom for the results in a diagram. Bubble Chart can only use value results. These are taken as values for the third (the z-axis). The bigger the value the greater the size of the corresponding bubble. The other two axis are filled with the DoF-(Degree of Freedom)Values. As one might follow every value result received by e.g. DQL needs to contain the exact same DoF's. If that is true Bubble Chart is enabled. As additional controls two combo boxes for the selection of the DoF's for each axis were added. Also another Spinner is shown. Here the user can choose a parameter to enlarge his bubbles, if the values are too low or to big to be shown nice.

3.2.8 Heat Map

Heat Map is a type of representation pretty similar to Bubble Chart. The only difference is that Heat Maps can only be used for continuous values. So PAVO only enables Heat Maps if Bubble Chart is already supported and all values of the different DoF's only differ by a maximum of 1.0. The additional controls are the exact same, only the Spinner is not existent in the Heat Map button panel.

3.2.9 Histogram

The graphic rendition of a histogram is also implemented in PAVO. Here the frequency of the x values is shown. As default all values are put into 5 different bins. The user is able to enlarge the number of bins by using the control feature of the histogram part. As a maximum the number of x values is used. If the result received has "Frequency" as y axis description the y values are used as frequency for the corresponding x value.

3.2.10 Pie Chart

The last diagram type actually supported by PAVO is a Pie Chart. It is only enabled, if the values of each series can be added to 1 or 100. Another way is by getting multiple value results, which can also be added to 1 or 100.

3.3 Result Type

To make PAVO as independent as possible we designed a new result model for it. In order to use PAVO one just needs to convert his result into the PAVO result model. This result model consists out of 5 Java classes. One interface, one container for multiple results, which shall be shown in one tab panel in GUI, and three different types of single results. In the Java code the classes are named with the Postfix "PAVO" as the simple name is for example already used in Descartes Modeling Language (DML)/DQL.

3.3.1 Abstract Quantitative Result

The interface we designed was kept as simple as possible. Every result consists out of a queryable element, a metric and a statistic type as a name. Furthermore there is one string for each axis as description and an additional description, which can be used to name the series according to something for example degrees of freedom, which are not saved in the other variables. Additionally to that there is the original location of the result element, to support auto update for the plots. This feature also needs the last variable dots shown for intern managing.

3.3.2 Result Container

The result container is a class with a list of different results. As all these results need to be shown in a single tab in the GUI a new variable for the axis labels are needed. Furthermore a parameter for the metric is implemented. This name is used to label the tab in the tab bar. More informations are not needed in the container.

3.3.3 Series Result

The class series result exist to represent a complete series in the PAVO result type. These series consists out of a list of Points, which have an x and y value and a confidence interval. Furthermore parameters if the series is continuous or discrete and one for the name are introduced.

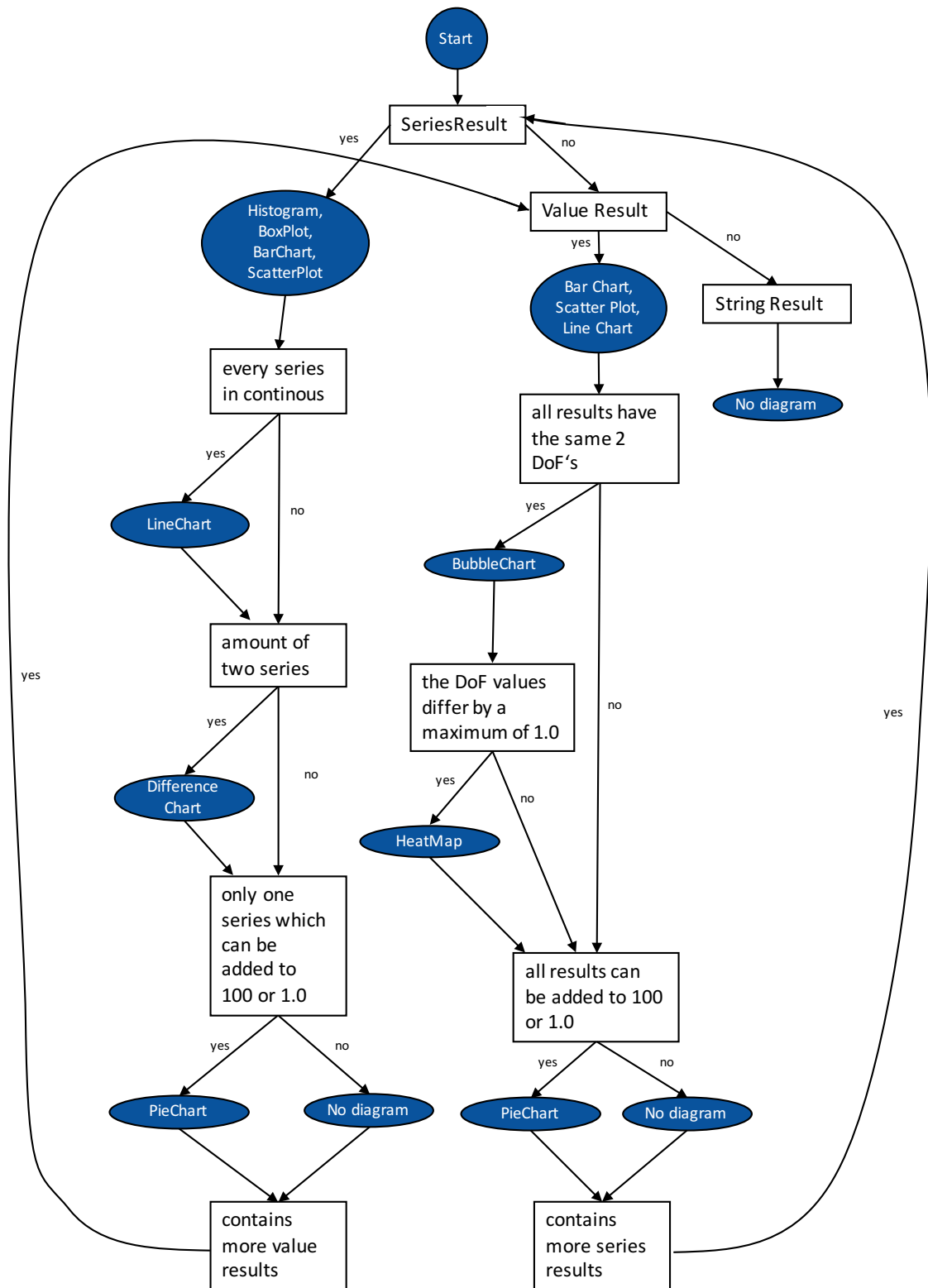


Figure 3.1: Decision diagram for the choice of diagram type

3.3.4 String Result

String values only have a string value as parameter which is later shown beneath the query string. No more values are needed.

3.3.5 Value Result

The value results are very similar to the string ones. Here also only one parameter for the integer value is implemented and nothing more. Results consisting out of a single value are shown as horizontal line in the plot.

3.4 Diagram Visualization

The main diagram visualization divides into two classes. The Graphics Engine, which manages the different tabs and the separation of the results given, and the Visualization Controller, which manages one single tab. The engine generates one controller for each tab that shall be created. At the current state of the library one tab for each metric is used. For more information about the classes, have a closer look to the next two subsections.

3.4.1 Graphics Engine

As already mentioned the Graphics Engine is the main class to start PAVO for your software. Create an object with a title and a list of EntityMappings. For the usage with other result types one needs to write a new constructor. This one shall work as the given two. With the only difference, that one also needs another method or class to convert this type into the PAVO type. For EntityMapping this is done in the TransformEntityMappingToResult class in the tools.descartes.PAVO.controller package. In addition to that another method is required to work with the tab separation one wants for its usage. Here you need to split your results into more result containers. (See Section 3.3.2). After that you create one VisualizationController (Section 3.4.2) for each and include that tab into the tabbed pane. An example for that is done in the interpretEntityMapping method.

3.4.2 Diagram Visualizer

The Visualizer is the most complex class of PAVO. It consists out of many different methods. Most of them use reflection to get the methods out of the current shown diagram type. To include another diagram type just implement the class and add it to the AddDiagramClasses list. All other methods are just for intern PAVO management and not necessary for the integration of PAVO into another software project. To get more information about them have a look to the JavaDoc of VisualizationController.java.

3.5 Diagram Export

If the user clicks on the export button the class ExportDialogFrame becomes active. It opens up a new frame with three parts. Two buttons to finish and cancel the progress. In addition to them there is a combo box which contains all export extensions currently used. Here the user is able to select his type of exported file. If this one is a picture, text boxes for the selection of the size show up. Here the user can insert his width and the height is adjusted automatically. The export of the diagrams shown in the PAVO GUI is implemented as an Extension Point to the PAVO package. All new extensions need to implement the Interface called IExportController, which is located in the PAVO package. This interface provides several methods. The export method, to save the current shown JFreeChart to the matching file, which is chosen before by the Export Dialog of PAVO.

One method called `updatePathToSuffix`, adds the corresponding file suffix to the chosen file, if it is not added by user. Two more methods are left: One for the String, shown in the combo box of the dialog, to choose the kind of export controller and another one which returns the identifier of this controller. In all cases these two methods shall return the same String.

In separated packages there are four different exporters implemented. One for comma separated values, one for semicolon separated values, one for export as png file and a last one for the vector graphics export. They are located in other packages in the DQL repository, as especially the vector graphics export and the pdf export use other libraries than the chosen JFreeChart. These are licensed under different licenses, which need to be accepted, if one wants to use PAVO with all its exports. If one wants to create another exporter for some kind of file type, it is only nessecary to create a new extension to the extension point and activate it.

4. Summary

Now we will shortly summarize the work done with PAVO in 2016. We designed an independent library, mainly matching to DQL, to visualize results given in charts. PAVO already provides many different diagram types, all with their own logic, if they can be used. PAVO contains an own result type, so other software packages only need to convert their results to that one to make PAVO work. As normally created diagrams are not only used for personal interests, we added an export feature to create several files, which can later be included into dissertations. PAVO was kept as simple as possible to extend by using the software pattern of Model-View-Controller and the Decorator pattern for the GUI. At the current moment PAVO is included into the DQL respository, which can be found under <https://se3.informatik.uni-wuerzburg.de/descartes/dql.git>. For the future PAVO can be extended with even more diagram types and shall be included into even more software packages, to simplify them in the idea of Declarative Performance Engineering.

5. Acronyms

DQL Descartes Query Language

DML Descartes Modeling Language

DPE Declarative Performance Engineering

QPME Queueing Petri net Modeling Environment

GUI Graphical User Interface

PAVO Performance Analyse VisualizatiOn