

Politecnico di Milano
Italy



University of L'Aquila
Italy

JMT
Java Modelling Tools

JMVA – JSIMwiz – JSIMgraph – JMCH – JABA – JWAT

user manual

May 12, 2026
for JMT version 1.4.0 and later

Contents

Contents iii

1 Introduction 1
1.1 A brief history of JMT . . . 2
1.2 How to install JMT . . . 2
1.3 Starting with the JMT suite . . . 3

2 JMVA – Exact and Approximate Solution Techniques 5
2.1 Introduction . . . 5
2.1.1 Starting the alphanumeric MVA solver . . . 5
2.2 Model definition . . . 5
2.2.1 Defining a new model . . . 6
2.2.2 Classes Tab . . . 7
2.2.3 Stations Tab . . . 7
2.2.4 Service Demands, Service Times and Visits Tabs . . . 9
2.2.5 Reference Stations Tab . . . 11
2.2.6 What-if Tab . . . 12
2.2.7 Comment *Tab* . . . 13
2.2.8 Expression Evaluator . . . 14
2.2.9 Model Solution . . . 14
2.2.10 Model Solution - What-if analysis . . . 15
2.2.11 Modification of a model . . . 18
2.3 AMVA - Approximate algorithms for closed models . . . 18
2.4 Menu entries . . . 20
2.4.1 File . . . 20
2.4.2 Action . . . 21
2.4.3 Help . . . 21
2.5 Examples . . . 21
2.5.1 Example 1 - A model with a single closed class . . . 21
2.5.2 Example 2 - A model with two open classes . . . 24
2.5.3 Example 3 - A model with a load dependent station . . . 26
2.5.4 Example 4 - A model with one open and one closed class . . . 27
2.5.5 Example 5 - Identification of the Optimal Population Mix with a two-class workload . . . 30
2.6 Command line for JMVA . . . 33
2.6.1 XML file format . . . 34

3 JSIMgraph (Simulation - Graphical interface) 39
3.1 Overview . . . 39
3.2 The home window . . . 40
3.3 The graphical interface . . . 41
3.4 Working with the graphical interfaces . . . 44
3.4.1 Defining a new model . . . 44
3.5 Classes of customers . . . 45
3.6 Distributions . . . 47
3.7 Performance indices . . . 55
3.7.1 Confidence Intervals . . . 58
3.7.2 Max Relative Error . . . 58
3.7.3 Advanced Metric Configuration . . . 58

3.8 Simulation Parameters . . . 59
3.9 What-If Analysis . . . 60
3.10 Finite Capacity Region (FCR) . . . 63
3.11 Type of Stations . . . 65
3.11.1 Source Station . . . 66
3.11.2 Sink Station . . . 68
3.11.3 Queueing Station . . . 68
3.11.4 Delay Station . . . 78
3.11.5 Fork Station . . . 80
3.11.6 Join Station . . . 83
3.11.7 Routing Station . . . 84
3.11.8 Logger Station . . . 85
3.11.9 Class-switch Station . . . 88
3.11.10 Semaphore Station . . . 91
3.11.11 Scaler Station . . . 92
3.11.12 Place and Transition Stations (Petri Nets) . . . 92
3.12 Modification of the Parameters . . . 98
3.12.1 Modifying simulation parameters . . . 98
3.12.2 Modifying station parameters . . . 99
3.12.3 Modifying the default values of parameters . . . 99
3.13 Error and Warning Messages . . . 103
3.14 Simulation results . . . 105
3.14.1 Results from a single simulation run . . . 105
3.14.2 Result of What-If Analysis simulation run . . . 106
3.14.3 Export to JMVA . . . 107
3.14.4 Transient detection . . . 107
3.14.5 Dead performance index . . . 109
3.15 Statistics on performance indexes . . . 111
3.15.1 How to obtain Statistics and Graphs of the simulation results . . . 111
3.15.2 Computation of the statistical values . . . 111
3.16 Examples . . . 115
3.16.1 Example 1 - A model with single closed class . . . 115
3.16.2 Example 2 - What-If analysis of a system with multiclass customers . . . 119
3.16.3 Example 3 - Using FCR for modelling bandwidth contention . . . 122
3.16.4 Example 4 - Closed models with class-switch . . . 123
3.17 jSIM Log . . . 124
3.18 Command line for JSIM*graph* . . . 124
3.19 The XML format of JSIM models . . . 133
3.20 The JSIMgraph Templates . . . 136
3.20.1 What are the *Templates* . . . 136
3.20.2 Download of *Official* Templates from JMT `sourceforge` site . . . 136

4 JSIM*wiz* (Simulation - Textual interface) 139
4.1 Overview . . . 139
4.1.1 Starting the discrete-event simulator . . . 140
4.2 Defining a new model . . . 140
4.2.1 Define Classes . . . 141
4.2.2 Define Stations . . . 143
4.2.3 Define Connections . . . 153
4.2.4 Station Parameters . . . 154
4.2.5 Define Performance Indices . . . 155
4.2.6 Reference Stations . . . 156
4.2.7 Finite Capacity Regions . . . 156
4.2.8 Define Simulation Parameters . . . 157
4.2.9 Define what if analysis . . . 158
4.3 Import in JMVA . . . 160
4.4 Modify default parameters . . . 161
4.5 Modify the Current Model . . . 164
4.6 Error and warning messages . . . 164

4.7 Simulation Results . . . 166
4.8 Start Simulation . . . 166

5 JMCH (Modelling Classroom Helper) 169
5.1 Introduction . . . 169
5.2 The Scheduling algorithms . . . 170
5.3 The Routing algorithms . . . 172
5.4 The graphical animation mechanism . . . 173
5.5 Results of the Animated simulations . . . 174
5.6 Markov Chain animation . . . 175
5.6.1 Starting the simulation . . . 175
5.6.2 Input parameters . . . 176
5.6.3 Output results . . . 177
5.6.4 Logger . . . 178

6 JABA (Asymptotic Bound Analysis) 179
6.1 Overview . . . 179
6.1.1 Starting the alphanumeric JABA solver . . . 179
6.2 Model definition . . . 179
6.2.1 Defining a new model . . . 180
6.2.2 Classes Tab . . . 181
6.2.3 Stations Tab . . . 181
6.2.4 Service Demands, Service Times and Visits Tabs . . . 181
6.2.5 Comment Tab . . . 183
6.2.6 Saturation Sector - Graph . . . 183
6.2.7 Convex Hull - Graph . . . 184
6.2.8 Utilization - Graph . . . 187
6.2.9 All In One - graph . . . 188
6.2.10 Saturation Sector - Text . . . 189
6.2.11 Modification of a model . . . 189
6.3 Menu entries . . . 189
6.3.1 File . . . 189
6.3.2 Action . . . 190
6.3.3 Help . . . 190
6.4 Examples . . . 190
6.4.1 Example 1 - A two class model . . . 190
6.4.2 Example 2 - A three class model . . . 193

7 JWAT - Workload Analyzer Tool 197
7.1 Workload analysis . . . 199
7.1.1 A workload characterization study . . . 200
7.1.2 Input definition . . . 200
7.1.3 Format definition . . . 204
7.1.4 Data processing . . . 207
7.1.5 Clustering algorithms . . . 210
7.1.6 The results of a clustering execution . . . 213
7.1.7 An example of a web log analysis . . . 218
7.1.8 Menus description . . . 221
7.1.9 Demo . . . 221
7.2 Fitting . . . 221
7.2.1 Input definition . . . 223
7.2.2 Pareto and Exponential panels . . . 223
7.2.3 Demo . . . 223

A Basic Definitions 227

B List of Symbols 229

C Acknowledgement 231

Bibliography **233**

Index **237**

Chapter 1

Introduction

Java Modelling Tools (JMT) is a free open source suite consisting of *six tools* for performance evaluation, capacity planning, workload characterization, and modelling of computer and communication systems. The suite implements several state-of-the-art algorithms for the exact, approximate, asymptotic and simulative analysis of queueing network models, either with or without product-form solution, integrated with some Petri Nets features. Multiformalism models are accepted. Models can be described either through *wizard* dialogs or with a *graphical* user-friendly interface. The workload analysis tool is based on clustering techniques. The suite uses an XML data layer that enables full reusability of the computational engines.

The JMT suite is composed by the following tools:

JSIM*wiz*: a wizard-based interface for the discrete-event simulator JSIM for the analysis of queueing network and Petri net models. A sequence of *wizard* windows helps in the definition of the network properties. The JSIM simulation engine supports several probability distributions for characterizing service and inter-arrival times. Load-dependent strategies using arbitrary functions of the current queue-length can be specified. JSIM*wiz* supports state-independent routing strategies, e.g., Markovian or round robin, as well as state-dependent strategies, e.g., routing to the server with minimum utilization, or with the shortest response time or with minimum queue-length, and load dependent routing. The simulation engine supports several extended features not allowed in product-form models, namely, finite capacity regions (blocking), fork-join servers (parallelism), class switching, priority classes and place-transition structures (synchronization). The JSIM performs automatically the transient detection, based on spectral analysis, computes and plots on-line the estimated values within the confidence intervals. What-if analyses, where a sequence of simulations is run for different values of control parameters, are also supported.

JSIM*graph*: a *graphical* user-friendly interface for the same simulator engine JSIM used by JSIM*wiz*. It integrates the same functionalities of JSIM*wiz* with an intuitive graphical workspace. This allows an easy description of network structure, as well as a simplified definition of the input and execution parameters. Network topologies can be exported in vectorial or raster image formats.

JMVA: for the *exact* and *approximate* analysis of single-class or multiclass product-form queueing networks, processing *open, closed* or *mixed* workloads. Several *exact* algorithms are available for *closed* networks: MVA (Mean-Value Analysis), RECAL (Recursion by Chain Algorithm), MoM (Method of Moments), CoMoM (Class-Oriented Method of Moments). A stabilized version of the MVA algorithm is used. Several *approximate* solution algorithms are also available (Chow, Bard-Schweitzer, AQL, Linearizer, De Souza-Muntz Linearizer). Network structure is specified by textual *wizards*. What-if analyses and graphical representation of the results are provided.

JMCH: modelling classroom helper that applies a simulation technique to solve a single-station model, including single-server stations with finite (M/M/1/k) or infinite (M/M/1) queue size, and multiple-server stations with unlimited (M/M/c) or limited (M/M/c/k) queue size. The animation of the underlying *Markov Chain* is also shown. It is possible to dynamically change the arrival rate, the service time and the queue size of the system.

JABA: for the identification of *bottlenecks* in multiclass closed product-form networks using efficient convex hull algorithms. Up to three customer classes are supported. It is possible to identify potential bottlenecks corresponding to the different mixes of customer classes in execution. Models with thousands of queues can be analyzed efficiently. *Optimization* studies (e.g., throughput maximization, minimization of response time,

identification of the optimal load) can be performed through the identification of the *saturation sectors*, i.e., the mixes of customer classes in execution that saturate more than one resource simultaneously.

JWAT: supports the *workload characterization* process. Some standard formats for input file are provided (e.g., Apache HTTP and IIS log files), customized formats may also be specified. The imported data can initially be analyzed using descriptive statistical techniques (e.g, means, correlations, histograms, boxplots, scatterplots), either for univariate or multivariate data. Algorithms for data scaling, sample extraction, outlier filtering, k-means and fuzzy k-means clustering for identifying similarities in the input data are provided. These techniques allow the identification of cluster of customers having similar characteristics. The clusters centroids represent the mean values of the parameters of the classes (e.g., CPU time, n.o of I/Os, n.o of web pages pages accessed) that can be used for the workload parameterization.

1.1 A brief history of JMT

JMT started in 2002 at Politecnico di Milano as a project led by Prof. Giuseppe Serazzi to extend an existing set of tools for performance evaluation called Windows Modelling Tools (WMT). The first MSc thesis developed a MVA tool and dated 1990. Initially, C was used as a programming language, then C^{++}, and since 2002 all the code was converted to Java and the JMT project was started. Fortran was used originally for coding the clustering algorithms K-means and Fuzzy K-means of the JWAT tool (1983) and the code was later migrated to Java in 2006.
On November 2013, Imperial College London joined Politecnico di Milano in managing the project and the development of new features. Tens of students and several research assistants were involved in the project while studying or working at Politecnico di Milano, Italy, and at Imperial College London, UK. On June 2025, the University of L'Aquila - Spencer Lab replaced Imperial College London in the management and development of the JMT.
At present, JMT consists of six tools JSIMwiz, JSIMgraph, JMVA, JMCH, JABA and JWAT. Since 2006, JMT has been downloaded more than 100561 times (May 12, 2026) by researchers, teachers, students and practitioners of 161 countries.
We are still planning new features and upgrades! We hope you will enjoy using it.

1.2 How to install JMT

You can install all the components of the *Java Modelling Tools* (JMT) downloading them free from `http://jmt.sourceforge.net/Download.html`. JMT is *free* software; it can be used under the terms of the GNU General Public License either version 2 of the License, or any later version.

It is recommended to uninstall any previous version of JMT before installing a new release.

The FULL Version includes all the JMT tools, the graphical installer, the user manual, examples, shortcuts. JMT is platform-independent and requires only the Oracle/Sun Java Runtime Environment (version 1.6 or later) to be installed on the machine. The Java Runtime Environment may be downloaded from `http://www.oracle.com/technetwork/java/javase/downloads/index.html` .

All required libraries are included in the download packages.
JMT has been successfully tested with Windows, Linux, and Mac OS. The user may select the **directory** in which **install** the JMT.

Another **new** folder is created in the home directory of the user for the examples and for some working files of JMT. Also the files created by the `Loggers` (*xxx*`.csv` files) are located in this folder. The location of the **JMT working folder** depends on the operating system of the machine and the user name:

- on **Linux** and **Unix** os in general, the user home is located in path `/home/<`*user name*`>/JMT`
- on **Windows** the folder is `C:\Users\<`*user name*`>\JMT`
- on **Mac OS X** the folder is `/Users/<`*user name*`>/JMT`.

Two new subdirectories are created in this folder: `JMT\examples` and `JMT\jwatFormat`. The subdirectory `\example` contains some models implemented with the JMT tools. The subdirectory `\jwatFormat` contains the description of the formats (in Perl5) of the data that are supplied as input for the demo of JWAT tool.

This manual and several other documents can be found in the sourceforge page of JMT.

1.3 Starting with the JMT suite

Double click on the JMT icon on your *program group* or on the *desktop*, or open the *command prompt* and type from the installation directory:

```
java -jar JMT.jar
```

The window of Figure 1.1 will appear. This starting screen is used to select the application of the suite to be

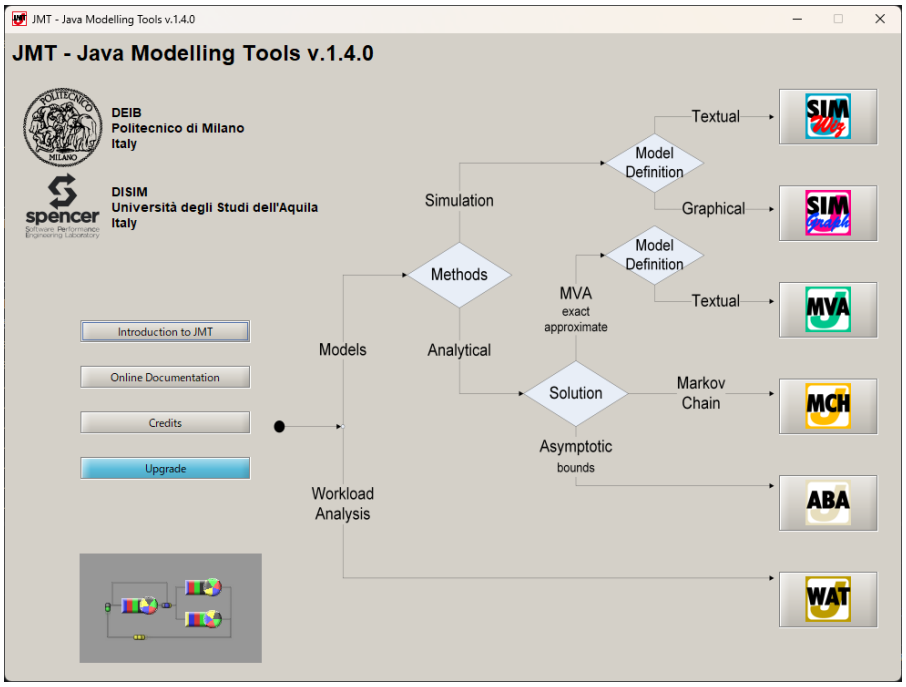


Figure 1.1: The JMT suite Starting Screen

executed by clicking on the corresponding button. The flow chart should help the user to select the application that best fits its needs.

In the following chapters the tools will be examined in details and some examples are given. This manual is intended for the general user that wants to learn how to interact with JMT. Advanced users that want to learn details on internal data structures, computational engines and XML interfaces should refer to *JMT system manual*.

Several other documents related to JMT description and applications are provided with the suite. Click on `Online Documentation` button to access the library.

Chapter 2

JMVA – Exact and Approximate Solution Techniques

2.1 Introduction

JMVA solves open, closed and mixed product form queuing networks [BCMP75]. Several solution algorithms are available. Open models are solved applying the classical queuing networks algorithms (QN), see e.g. [Kle75] [Ste09][Tri02]. The *exact* solution algorithms available for closed queuing networks are: MVA (Mean-Value Analysis) [RL80], RECAL (Recursion by Chain Algorithm) [CG86], MoM (Method of Moments) [Cas06] [Cas11], CoMoM (Class-Oriented Method of Moments) [Cas09], TreeMVA [TS85]. In order to avoid numerical instabilities of the solutions when the model contains load dependent stations, the implemented exact MVA algorithm is a *stabilized* version [BBC$^+$81] of the classic MVA version. The *approximate* AMVA algorithms available are: Chow [Cho83], Bard-Schweitzer [Sch79], Linearizer [CN82], DeSouza-Muntz Linearizer [dSeSM90], Aggregate Queue Length (AQL) [ZES88] .
The use of some of the exact and approximate solution algorithms is motivated by the *very large* time and space requirements of the exact MVA algorithm when the models have large dimensions (high number of classes, of resources, and of customers), a case commonly encountered in modern web and cloud applications.

Resources may be of two types: *queueing* (either with `load independent` or `load dependent` service times) and `delay`. The model is described in alphanumeric way: user is guided through the definition process by steps of a *wizard* interface. *What-if* analyses, where a sequence of model is solved for different values of the parameters, are also possible (see subsection 2.2.6). The solutions provided by several approximate algorithms and by the exact MVA can be compared through suitable graphical representation.

A graphical interface to describe the model in a user-friendly environment is also available, see JSIM*graph* for details.

2.1.1 Starting the alphanumeric MVA solver

Selecting  button on the starting screen, Figure 2.1 window shows up. Three main areas are shown:

Menu : it is organized into three groups of functions. To use a menu, click on the menu heading and choose the appropriate option. For the description of menu entries, see section 2.4

Toolbar : it contains some buttons to speed up access to JMVA functions (e.g. New model, Open, Save... See section 2.4 for details). If you move the mouse pointer over a button a tooltip will be shown up.

Page Area : this is the core of the window. All MVA parameters are grouped in different tabs. You can modify only a subset of them by selecting the right tab, as will be shown later.

2.2 Model definition

Models with one or multiple customer classes provide estimates of performance measures. For a brief description of basic terminology please refer to Appendix A.

In the case of single class models, the workload is characterized by two inputs: the set of service demands, one for each resource, and the workload intensity. On the other hand, in multiple class models, a matrix of service demands is requested [LZGS84].

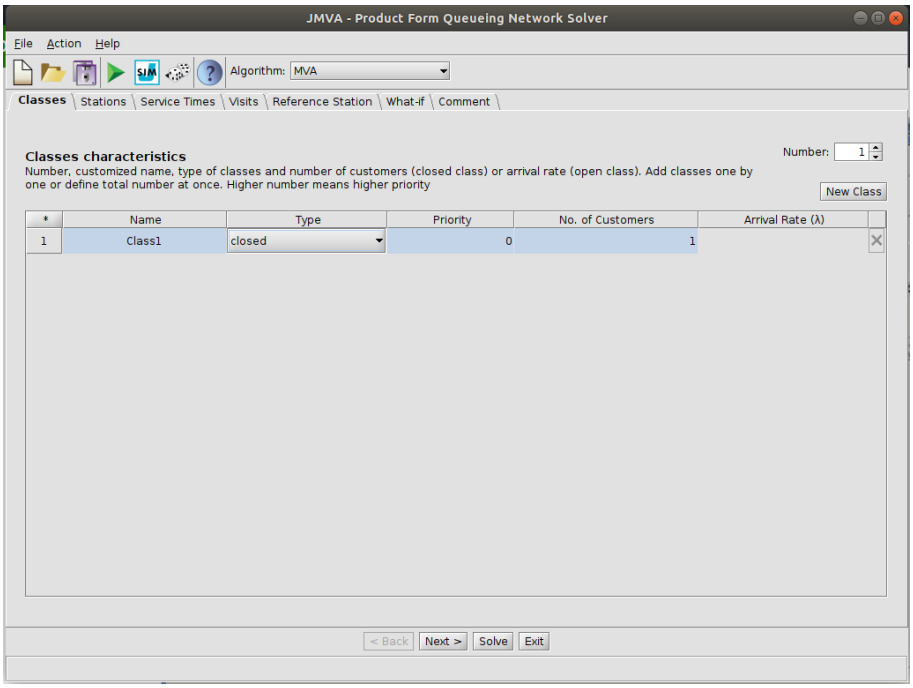


Figure 2.1: Classes tab

2.2.1 Defining a new model

To define a new model select toolbar button or the *New* command from *File* menu. The following parameters must be defined:

1. `Classes` with their workload intensities (number of customer N for closed classes and arrival rate λ for open classes)
2. `Stations` (service centers)
3. `Service times` and `Visits` (or `Service demands`)
4. `Reference Stations`
5. `What-if`
6. Optional short `Comment`

The execution of JMVA provides, for each class and each station, the following performance indices:

- Throughput
- Queue Length
- Residence Time
- Utilization
- System Power (throughput/response time for *each class*) [Kle79]

The following *aggregate* indices are provided:

- System Throughput
- System Response Time
- Average number of customers in the system
- System Power: Aggregate System Throughput (i.e., sum of the per-class throughputs) / Aggregate System Response Time (i.e., sum of the per-class response times weighted by the relative throughputs)

Input tabs

As can be seen in Figure 2.1, the parameters that must be entered in order to define a new model are divided in the following tabs: `Classes`, `Stations`, `Service times`, `Visits`, (or `Service Demands`), `Reference Stations`, `What-if`, and `Comment`.

You may input the `Service Times` and `Visits` for the stations, if you know, or alternatively the `Service Demands` (that are the products of `Service times` and `Visits`). As a function of your choice, the `Service Demands` *Tab* or the `Service Times` and `Visits` *Tab*s will be hidden, see subsection 2.2.4. You can navigate through tabs:

- using sequential wizard buttons, if enabled, at the bottom of the window (Figure 2.2)
- using sequential buttons located in menu
- using the tab selector, clicking on the corresponding tab (Figure 2.3)

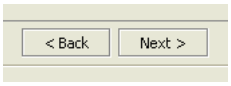


Figure 2.2: Wizard buttons

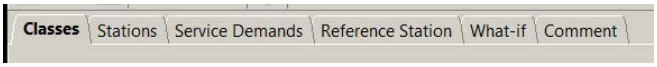


Figure 2.3: Tab selector

2.2.2 Classes Tab

An example screenshot of this tab can be seen in Figure 2.1. This tab allows to characterize customer classes of the model. Your model will be a single class model if and only if there will be only one class, closed or open. On the contrary multiple class models will have at least two classes, closed and/or open.

The number of classes in the model can be specified in the corresponding input area, shown in Figure 2.4 and can be modified using the keyboard or using the spin controls.

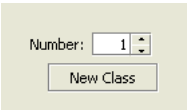


Figure 2.4: Number of classes

Using the delete button  associated to a specific class, a class can be removed provided that there will be at least one class after the deletion. Similar result may be obtained using spin controls, decreasing classes number; in this case last inserted class will be removed.

Default class names are *Class1*, *Class2*, ... *ClassR*. A model can be customized by changing these names: this can be done by clicking directly on the name that should be changed.

In Figure 2.5 there are three classes of customers, two closed and one open. The third class has the default name *Class3* while the other two classes have customized names, namely *ClosedClass* and *OpenClass*.

A class type can be `Open` or `Closed`. It is important to define each class type because a closed class workload is described by a constant number of customers in each class, while the open classes workload is described by the customer arrival rate for each class which does not imply limits on the maximum number of customers.

As can be seen in Figure 2.5, a class type can be selected in a combo-box. The input boxes *No. of Customers (N)* referring to closed classes accept only positive integer numbers; the input boxes of the *Arrival Rate* (λ) referring to open classes, accepting positive real numbers (Figure 2.6).

The *Priority* column is by default set to 0 to indicate no priority. If increased to a positive integer for a class, it will give more priority to that class. Larger priority values mean that the higher priority classes will receive service with precedence over classes with lower priority values. Identical priority values mean that the classes will receive equal treatment at stations.

ATTENTION: In order to consider priorities at a given station, a *Station Type* such as `FCFS Preemptive Resume` or `FCFS Non-Preemptive Head-of-Line` must be selected under the *Stations* panel.

2.2.3 Stations Tab

The number of stations of the model can be specified in the corresponding input area (Figure 2.7) and can be modified using the keyboard or the spin controls.

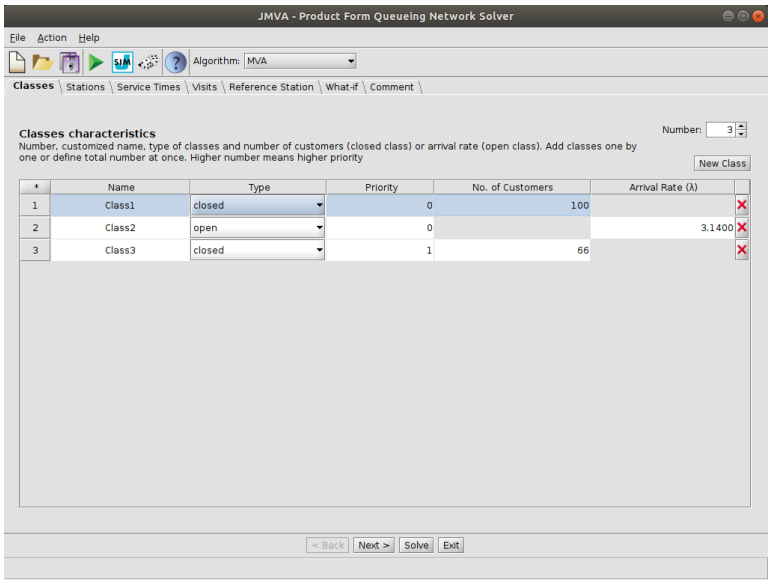


Figure 2.5: Defining the classes types



Figure 2.6: Workload definition of the number of customers of a closed class ($N = 100$) and the arrival rate of an open class ($\lambda = 3.14$)

Using the delete button  associated to a specific station, a station can be removed provided that there will be at least one station after the deletion. A similar result may be obtained using spin controls, decreasing stations number; in this case last inserted station will be removed.

Default station names are *Station1*, *Station2*, ... *StationM*. In order to personalize your model, you can change and give names other than default ones.

In Figure 2.8 there is only one station with default name *Station4* and there are three stations with customized names: *CPU*, *Disk1* and *Disk2*.

A station type can be:

- `Load Independent`. A regular queueing station with a finite number of servers.
- `Load Dependent`. `Load Dependent` stations are **not supported** for **open class** workloads.
- `Delay`. A queueing station with an infinite number of servers, used to model pure delays with no queueing.
- `Load Independent: FCFS Preemptive Resume`. A load-independent station that processes priority according to a preemptive resume policy. That is, upon arrival of a job, the job in service will be preempted and resumed later.
- `Load Independent: FCFS Non-Preemptive Head-of-Line`. A load-independent station that, upon arrival of a job with higher priority, still continues to finish the job in service irrespectively of its priority value. Priorities will be used only at the instant of completion of the job in service to select the next job to execute.

It is important to define each station `type` because if a station is `Load Dependent` a set of `service demand` - or a set of `service times` and `number of visits` - must be defined (one service demand/time for each possible value of queue length inside the station). In subsection 2.2.4 we will explain this concept with more details.

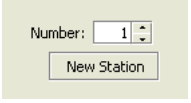


Figure 2.7: Number of stations

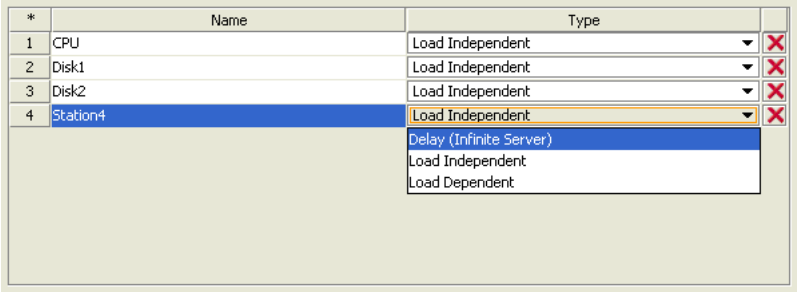


Figure 2.8: Defining the stations type

2.2.4 Service Demands, Service Times and Visits Tabs

Service Demands can be defined in two ways:

- directly, by entering Service Demands (D_{kc})
- indirectly, by entering Service Times (S_{kc}) and Visits (V_{kc})

Service demand D_{kc} is the total service requirement, that is the average amount of time that a customer of class c spends in service at station k during one interaction with the system, i.e. it completes execution. Service time S_{kc} is the average *execution* time spent by a customer of class c at station k for a single visit at that station (it *does not include* the waiting time in queue!) while V_{kc} is the average number of visits at that station during a complete execution of the job.

Remember that it is $D_{kc} = V_{kc} * S_{kc}$ so it is simple to compute the service demands matrix starting from service times and visits matrices. Inverse calculation is performed with the following algorithm:

$$V_{kj} = \begin{cases} 1 & \text{if} \quad D_{kc} > 0 \\ 0 & \text{if} \quad D_{kc} = 0 \end{cases}$$

$$S_{kc} = \begin{cases} D_{kc} & \text{if} \quad D_{kc} > 0 \\ 0 & \text{if} \quad D_{kc} = 0 \end{cases}$$

ATTENTION: if you define a model with the parameters `Service Times` and `Visits`, and later you will use the `Service Demands`, the values of the `Visits` will be lost (because, as above explained, they are set to 1).

Service Demands Tab

In this tab, you can insert directly the `Service Demands` D_{kc} for each pair {station k, class c}. In Figure 2.9 a reference screenshot can be seen: notice that the values of the D_{kc} elements of the D-matrix have already been specified because the *default* value assigned to *newly* created stations is *zero*.

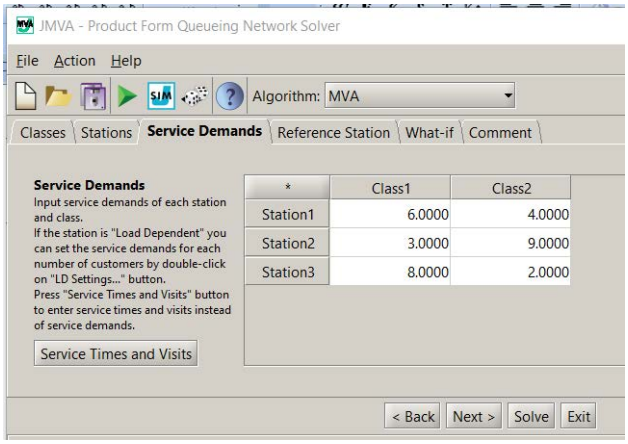


Figure 2.9: Example of the `Service Demands` Tab

In the example of Figure 2.9, each job of type *ClosedClass* requires an average service demand time of 6 sec to *CPU*, 10 sec to *Disk1*, 8 sec to *Disk2* and 2.5 sec to *Station4*. On the other hand, a job of type *OpenClass*

requires on average 0.1 sec of *CPU* time, 0.3 sec of *Disk1* time, 0.2 sec of *Disk2* time and 0.15 sec of *Station4* time to be processed by the system.

If the model contains any load dependent station, the behavior of Figure 2.10 will be shown. By double-

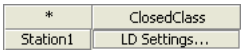


Figure 2.10: Defining a *load dependent* station service demand

clicking on *LD Settings...* button a window will show up and that can be used to insert the values of the service demands for each possible number of customer inside the station. That values can be computed by evaluating an analytic function as shown in Figure 2.11. The list of supported operators and more details are reported in subsection 2.2.8.

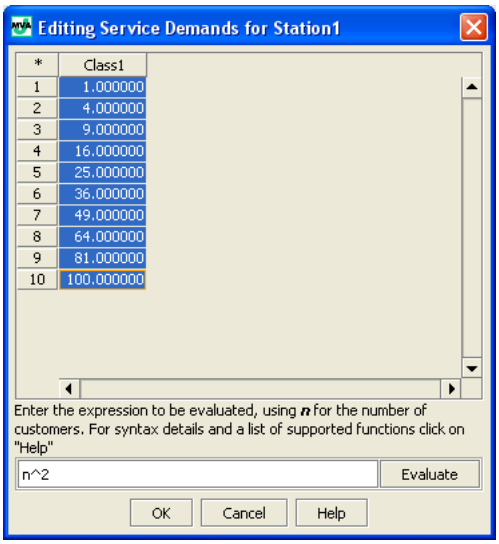


Figure 2.11: Load Dependent editing window

Service Times and Visits Tabs

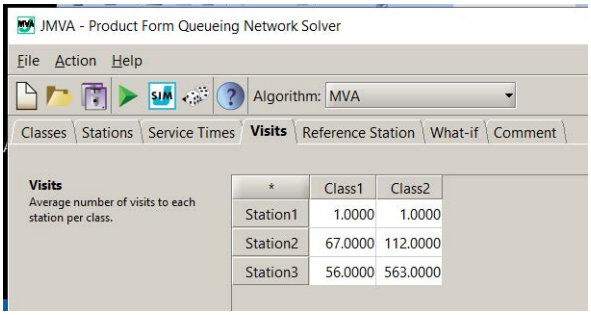


Figure 2.12: Visits Tab

In the former tab you can insert the `Service Times` S_{kc} for each pair {station k-class c} in the model, in the latter you can enter the visit count V_{kc} (See Figure 2.12).

The layout of these tabs is similar to the one of the `Service Demand Tab` described in the previous paragraph. The default value for each element of the `Service Times` (S) matrix is zero, while it is one for `Visits`' matrix elements.

If current model contains load dependent stations, the behavior of `Service Times Tab` for their parameterization will be identical to the one described on the previous paragraph for `Service Demands Tab`. On the other hand `Visits Tab` behavior will not change as load dependency is a property of service times and not of visits.

2.2.5 Reference Stations Tab

The concept of `Reference Station` is often obscure, and sometimes doubts arise about the actual need of its introduction. A user may argue that s/he may define and solve a model without knowing the existence of such a station. S/he may think that it is sufficient to parameterize the workload (type of classes and intensity), the Service Times and Visits, the Stations, and then select a solution algorithm. However, the knowledge of these parameters implies several assumption that were made *implicitly*:

- the user knows the number of times that a job visit each station during its complete execution
- the existence of a station external to the model that jobs leave when submitted for execution and join when their execution is completed and exit the model
- this external station, that may be explicitly represented in the model (typically is a delay station representing the users) or assumed existing by default, by definition is *visited only once* by the jobs that have completed their execution
- this external station, referred to as `Reference Station`, is used to measure the *throughput* X and *response time* R (or *cycle time*) at the *system level*

However, depending on the problem to be solved, it is sometimes necessary to adopt a different level of detail for the analysis of performance than the one of the entire system (typically represented by the Users Station). So, you may need to consider a `Reference Station` different from the one defined above (external to the system) that could have a number of visits greater than one. To compute the correct values of throughput and response time with respect to this new station it is necessary to scale the value of the visits of all the stations of the model in order to made the visits to it equal to one.

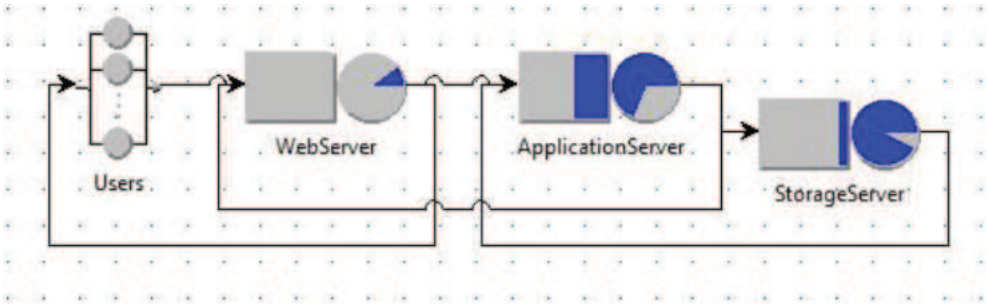


Figure 2.13: Example of a three tier intranet with a `Reference Station` representing the *Users*

In the model of Fig.2.13 is reported a three tier intranet in which the Reference Station is explicitly represented through the Users station (a delay station). In Fig.2.14 is shown the selection of the `Users` station

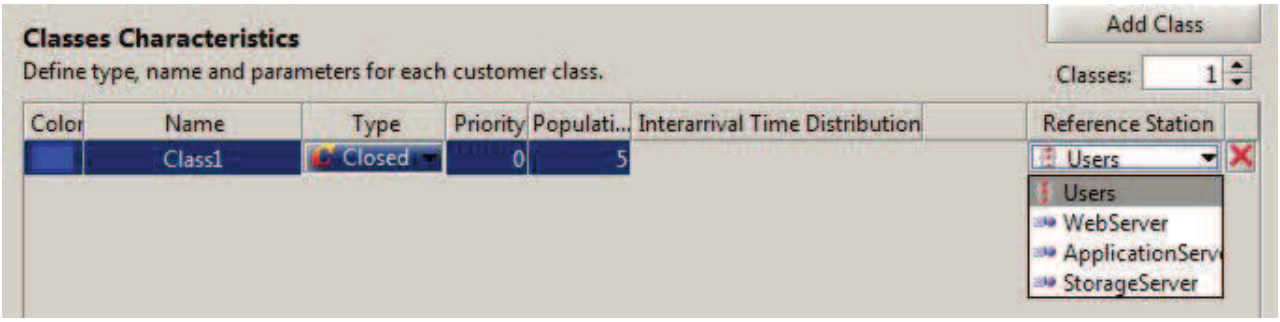


Figure 2.14: Selection of the `Users` station as `Reference Station`

as `Reference Station`. The corresponding `System Response Time` is 97.055 sec. A job during its complete execution visit the `Web Server` 10 times ($V_{WebServer} = 10$). If the problem requires the computation of the `System Response Time` with respect to the `Web Server` (i.e., the time between two consecutive passages in the `Web Server` by the same job) with the scaling of the visits the new value of `System Response Time` will be 9.751 sec (see Fig.2.16).

WARNINGS:

— if the workload consists of multiple *closed classes* **all** the classes must have the *same* `Reference Station`

— the *visits* to the `Reference Station` of *all* the classes must be different from 0

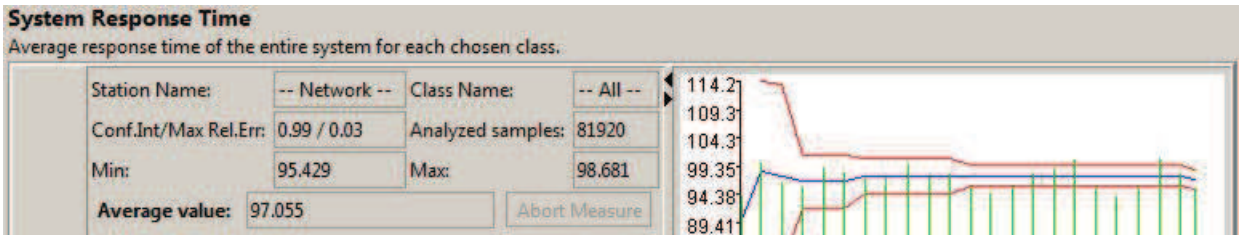


Figure 2.15: `System Response Time` of the model of Fig.2.13 with the `Users` Station as `Reference Station`.

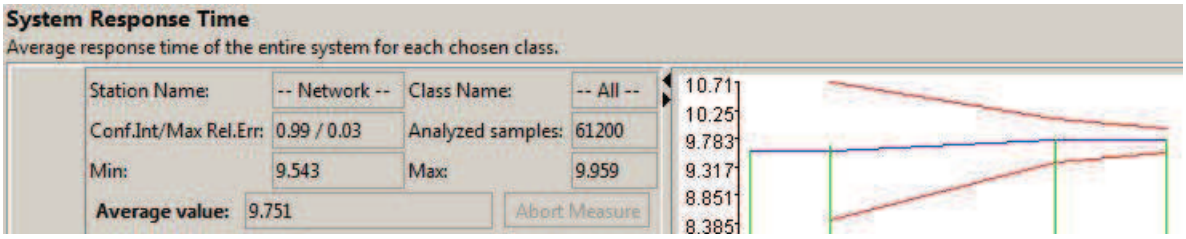


Figure 2.16: `System Response Time` of the model of Fig.2.13 with the `Web Server` Station as `Reference Station`.

— the `Reference Station` of *open classes* is selected by default on the `Source Station`
— if you parameterize the model with the `Service Demands` D_i, then it is not important which station is selected as `Reference` because all of them are visited only *once* with service times S_i equal to the D_i.

2.2.6 What-if Tab

This Tab is used to perform a `what-if` analysis, i.e., to solve multiple models with different values of a `control parameter`. Figure 2.17 shows the what-if panel when what-if analysis is disabled. In Figure 2.17 is shown this panel when what-if analysis is disabled.

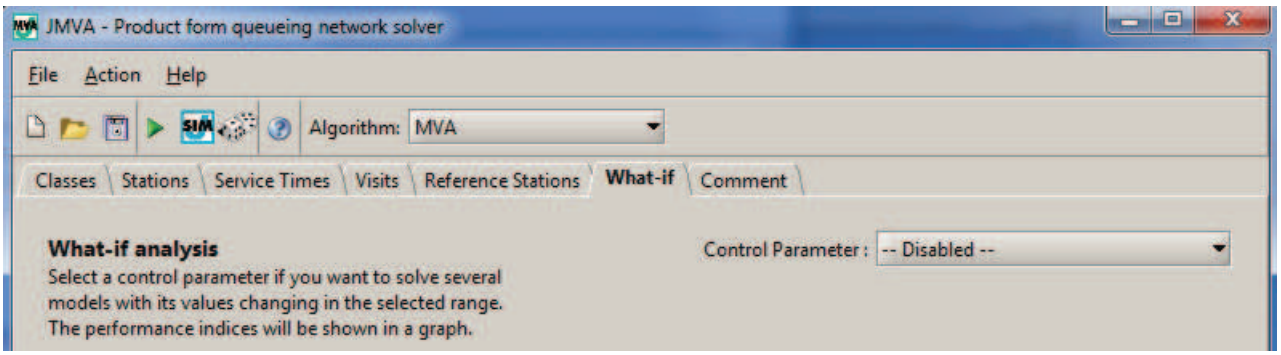


Figure 2.17: What-if Tab - to enable the analysis of several models

The first parameter to be set is the `control parameter` i.e. the parameter that will be changed to solve different models in a selected range. Five choices are possible:

Disabled : disables what-if analysis, so only a single queueing network model, specified in the previous steps, will be solved. This is the *default* option.

Number of Customers : different models will be executed by changing the `number of customers` of a single `closed` class or of every closed class proportionally. This option is available only when current model has at least one closed class.

Arrival Rates : different models will be executed by changing `arrival rate` of a single `open` class or of every open class proportionally. This option is available only when the current model has at least one open class.

Population Mix : the total number of customers will be kept constant, but the population mix (i.e. the ratio between `number of customers` of selected closed class i and the total number of customers in the system $\beta_i = N_i / \sum_k N_k$). This option is available only when the current model has two closed classes.

Service Demands : different models will be solved changing the `service demand` value of a given station for a given class or for all classes proportionally. This option is available only for `load independent` and

`delay` stations.

Whenever a control parameter is selected, the window layout will be changed to allow the selection of a valid range of values for it. For example in Figure 2.18, the `Service Demands` control parameter was selected. On the bottom of the window, a summarizing table is presented: depending on the selected control parameter, that table is used to show the *initial state* of the involved parameters. Every class currently selected for what-if analysis is shown in red.

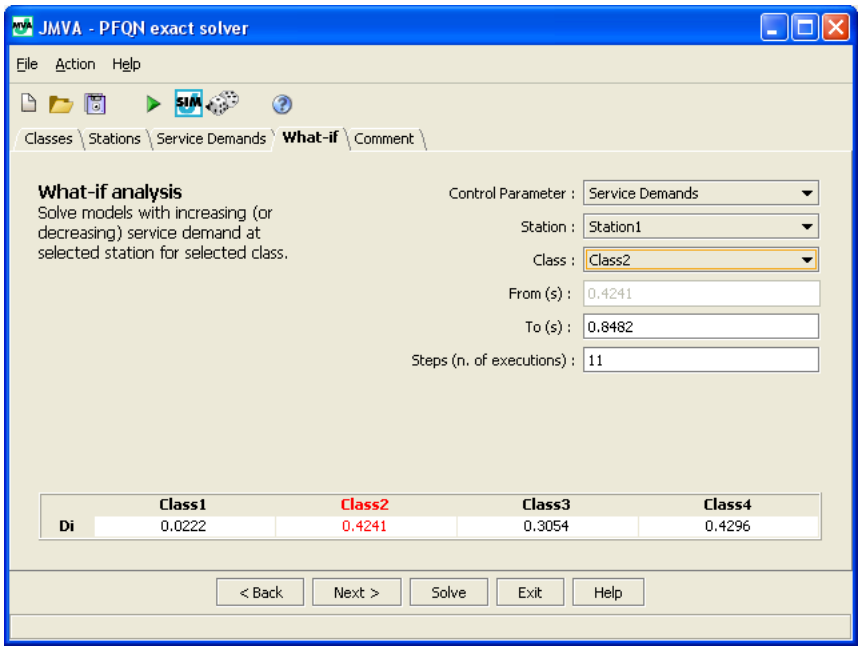


Figure 2.18: What-if Tab - Service Demands

A brief description of each field is now presented:

Station : available only with `Service Demands` control parameter. This combo box allows to select at which station service demand values will be modified.

Class : allows to select for which class the selected parameter will be changed. A special value, namely `All classes proportionally`, is used to modify the control parameter for each class keeping constant the proportion between different classes[1]. This special value is not available in `Population Mix` analysis as we are changing the proportion of jobs between two closed classes.

From : the initial value of what-if analysis. It was chosen to leave this value fixed to the initial value specified by the user in the previous steps to avoid confusions, so this field acts as a reminder. The only exception is when `Population Mix` is changed, in that case it is allowed to modify this value too.

To : the final value of what-if analysis. Please notice that this value can be greater or smaller than `From` value and is expressed in the same measure unit. Whenever `All classes proportionally` option is selected, both `From` and `To` values are expressed as percentages of initial values (specified in the previous steps and reminded in the table at the bottom of the panel, see Figure 2.18), in the other situations they are considered as absolute values for the chosen parameter.

Steps : this is the chosen number of executions i.e. the number of different models that will be solved. When control parameter is `Customer Numbers` or `Population Mix`, the model can be correctly specified only for integer values of population. JMVA will perform a fast computation to find the maximum allowed number of executions given current `From` and `To` values: if the user specifies a value bigger than that, JMVA will use the computed value.

2.2.7 Comment *Tab*

In this Tab, a short - optional - comment about the model can be inserted; it will be saved with the other model parameters.

[1] for example, in a model with two closed classes with population vector (2,6), the following models can be executed: (1,3), (2,6), (3,9), (4, 12), ...

2.2.8 Expression Evaluator

An expression evaluator is used for the definition of service demands or service times of a load dependent station. It allows to specify times as an analytic function of n where n is the number of customer inside the station.

Expression are evaluated using *JFEP*[2] (Java Fast Expression Parser) package which supports all operators enumerated in Table 2.1 and all functions enumerated in Table 2.2.

Operator	Symbol
Power	^
Unary Plus, Unary Minus	$+n$, $-n$
Modulus	%
Division	/
Multiplication	*
Addition, Subtraction	+, −

Table 2.1: List of all supported operators ordered by priority

Function	Symbol
Sine	sin()
Cosine	cos()
Tangent	tan()
Arc Sine	asin()
Arc Cosine	acos()
Arc Tangent	atan()
Hyperbolic Sine	sinh()
Hyperbolic Cosine	cosh()
Hyperbolic Tangent	tanh()
Inverse Hyperbolic Sine	asinh()
Inverse Hyperbolic Cosine	acosh()
Inverse Hyperbolic Tangent	atanh()
Exponential	exp()
Natural Logarithm	ln()
Logarithm base 10	log()
Absolute Value / Magnitude	abs()
Random number $[0, 1]$	rand()
Square Root	sqrt()
Sum	sum()

Table 2.2: List of supported functions for the load dependent service times

2.2.9 Model Solution

The window of Fig.2.19 allows the selection of the algorithm to be used to solve the models. If the model is open, standard queuing network algorithms are used (identified with QN). If the model is closed, several exact and approximate solution algorithms are proposed. Fig.2.19 shows the exact algorithms. Once selected the algorithm, use `Solve` command to solve the model. If the model specifies a what-if analysis, please refer to subsection 2.2.10. Model results will be shown on a separate window, like the one of Figure 2.20.

Using the tab selector, all the other computed performance indices can be seen: Throughput, Queue lengths, Residence Times, Utilizations and a synopsis panel with schematic report of input model. Both results and synopsis tab data can be copied to clipboard with the standard `CTRL+C` keyboard shortcut.

When open classes are used, the resource saturation control is performed. For multiple class models, the following inequality must be satisfied:

$$\max_k \sum_c \lambda_c * D_{kc} < 1$$

where c is the class index and k is the station index. This inequality ensures that no service center is saturated as a result of the combined loads of all the classes. Let us consider, as example, the model with the classes

[2] http://jfep.sourceforge.net/

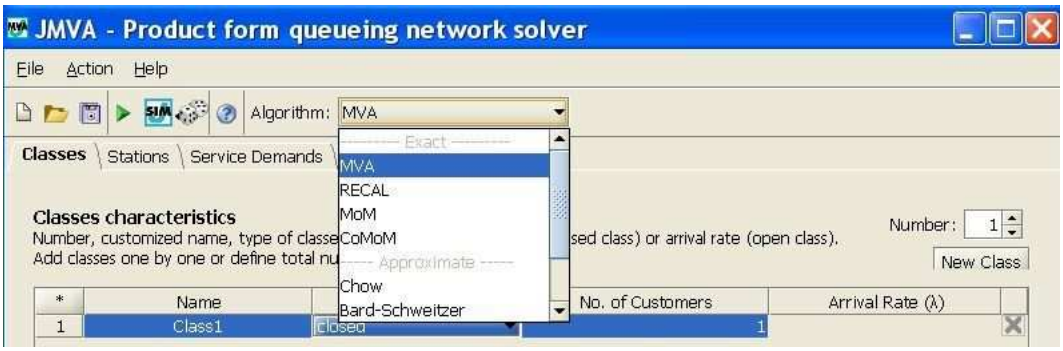


Figure 2.19: Drop-down box for the selection of the exact solution algorithm for closed classes

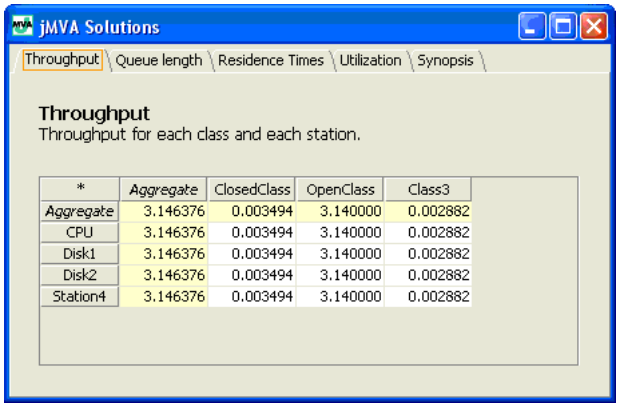


Figure 2.20: Model Solution (Throughput Tab)

shown in Figure 2.5 with the D-matrix shown in Figure 2.9 Since $\lambda = 3.14 < 3.33 = 1/0.3 = 1/D_{max}$ the model is not in saturation and the `Solve` command will be executed correctly.

In this example, substituting $D_{\text{Disk1-OpenClass}}$ with values $\geq 1/3.14 \approx 0.318$ will cause the saturation of resource *Disk1* and the error message of Figure 2.21 will appear.

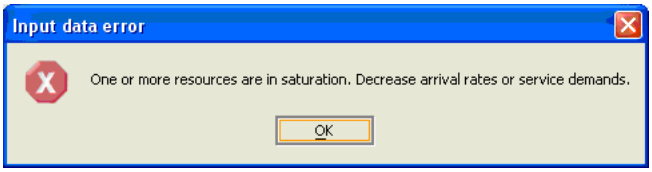


Figure 2.21: Input data error message

2.2.10 Model Solution - What-if analysis

Use the `Solve` command to solve the model. During model solution, a progress window, see Figure 2.22, shows up. It displays the cumulative number of models currently solved, the total number of models to be solved and the elapsed time.

At the end of the solution, results will be shown in a separate window, see Figure 2.23. This tab allows to show in a plot the relation between the chosen control parameter (see subsection 2.2.6) and the performance indices computed by the analytic engine.

The combo box `Performance Index` allows to select the performance index to be plotted in the graph, while in the table below, users can select the resource and the class considered.

- The first column is fixed and lists all available colors to be used in the graph.
- The second column, named `Class`, is used to select the class considered in the graph. The special value `Aggregate` is used for the aggregate measure for all classes. If input model is single-class, the class is selected by default for each row.
- The final column, named `Station`, is used to select the station considered in the graph. The special value `Aggregate` is used for the aggregate measure for the entire network. Note that the `Aggregate` value is not valid when the `Utilization` performance index is selected.

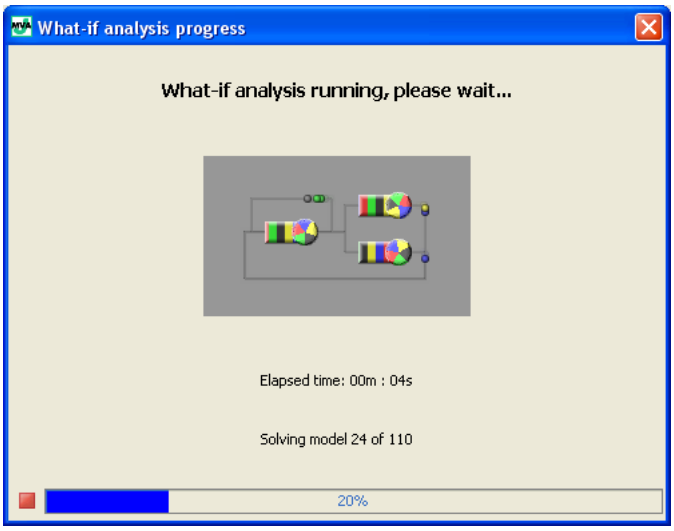


Figure 2.22: Model Solution progress window

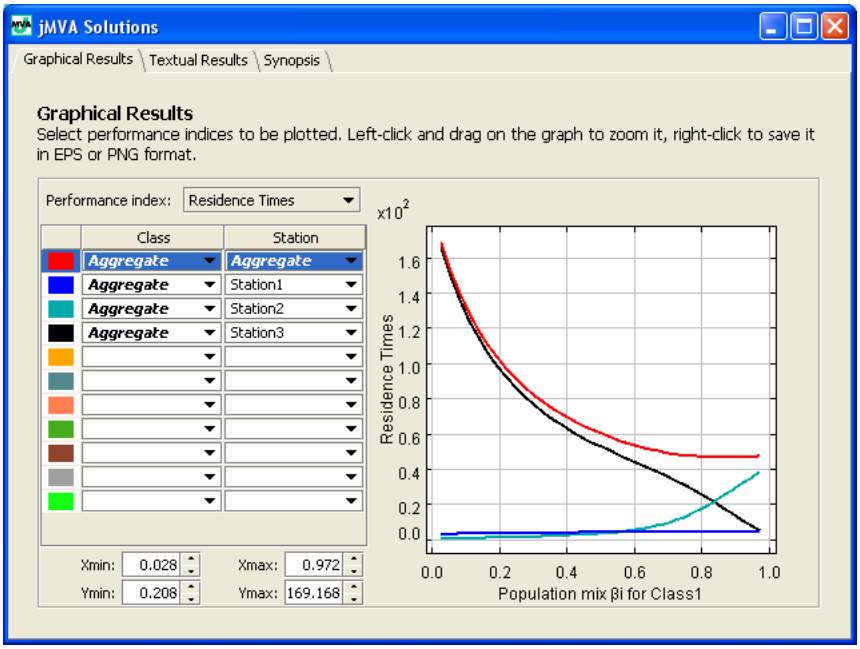


Figure 2.23: Model Solution - Graphical Results Tab

In addition to the *center* performance indices (i.e. `Throughput`, `Queue length`, `Residence Times`, `Utilization`), three *system* performance indices are provided in the `Performance Index` combo box (`System Response Time`, `System Throughput`, `Number of Customers`). This *system* indices can be easily obtained by selecting the special `Aggregate` value for both `Class` and `Station` columns of the corresponding center indices (see Appendix A for the definition of the performance indices), but they were provided here as *shortcuts*. As we are referring to aggregate measures, the selection of reference class and station is not significant and, in this case, the table in the left of Figure 2.23 will not be shown.

On the bottom-left corner of the window, users can modify minimum and maximum value of both the horizontal and vertical axes of the plot. JMVA is designed to automatically best-fit the plot in the window but this controls allow the user to specify a custom range or zoom on the plot. Another fast method to perform a zoom operation is to left-click and drag a rectangle on the graphic window (see Figure 2.24) or right-click on it and select `Zoom in` or `Zoom out` options. To automatically reset the best-fit scale users can right-click on the graphic window and select `Original view` option.

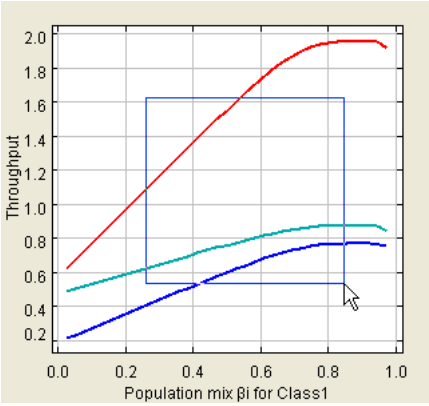


Figure 2.24: Zoom operation on the plot

The graphic window allows to export plots as image to be included in documents and presentations. To save current graph as image, right-click on the graphic and select `Save as...` option. A dialog will be shown to request the name of the file and the format. Currently supported format are Portable Network Graphics - PNG - (raster) and Encapsulated PostScript - EPS - (vectorial, currently only black and white).

The second tab of the solution window, shown in Figure 2.25, is used to display the solutions of each execution of the analytic algorithm.

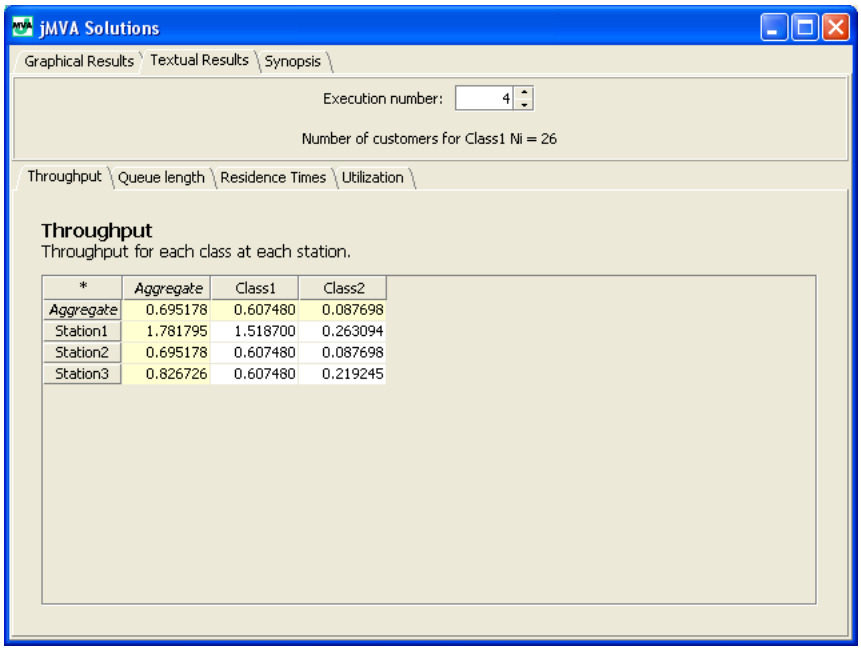


Figure 2.25: Model Solution - Textual Results Tab

This Tab has the same structure of the results window without What-if analysis (described in subsec-

tion 2.2.9) but allows to select the execution to be shown in the field `Execution Number`. By entering requested execution number in the spinner, or using the *up* and *down* arrows, user can cycle between all the computed performance indices for each execution. Just below the spinner, a label gives information on the value of the control parameter for the currently selected execution.

2.2.11 Modification of a model

To modify system parameters return to the main window and enter new data. After the modifications, if you use `Solve` command, a new window with model result will show. You can `save` this new model with the previous name - overwriting the previous one - or `save` it with a different name or in a different directory.

2.3 AMVA - Approximate algorithms for closed models

The exact MVA is an iterative algorithm. The solution of a closed model with N customers requires the values of indexes computed for the same model with N-1 customers. Thus, to obtain the performance indexes with N customers it is required to solve N models with customer number increasing from 0 to N. The key equation, with a single class workload, that introduces the iterative characteristic of MVA is:

$$A_i(N) = N_i(N-1) \tag{2.1}$$

that states that the number of customers $A_i(N)$ seen by a customer at arrival to station i in a closed model with N customers is equal to the mean number N_i of customers in station i of the same model with $N-1$ customers. As a consequence, the computation time and space required are non negligible, and become very very large with models having multiple class workloads, and high number of resources and customers.
Approximate algorithms recursively estimate station queue lengths for models with one less customer to compute statistics for models with full population until convergence is reached. The *stopping criterion* (referred to as `Tolerance` in the AMVA window, see, e.g., Fig.2.27) is based on the differences between successive queue lengths that must be less than the tolerance value. The AMVA algorithms replace eq.2.1 (or its versions for multiple class workloads) with *non-recursive* approximations of the type

$$A_i(N) \sim h[N_i(N)] \tag{2.2}$$

with suitable functions for h.

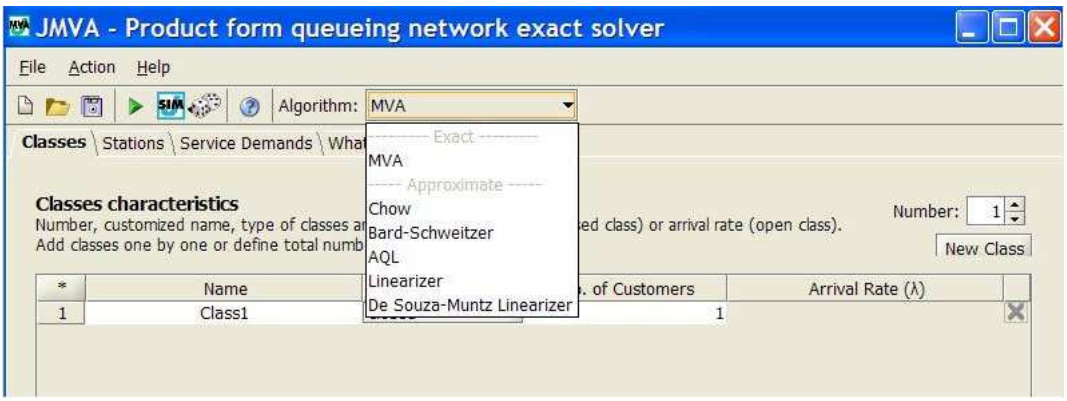


Figure 2.26: Drop-down box for the selection of the *approximate* solution algorithm

The approximate methods available are listed in the drop-down box of the JMVA window (see Fig.2.26). Different algorithms can be selected concurrently using the checkboxes shown in the `what-if` analysis window (see Fig.2.27) and their results can be compared directly. The graphics of the performance indexes generated by the algorithms selected in the checkboxes are shown in Fig.2.28.

For each class of customers c, the *input* parameters are the number of customers N_c,the service demands $D_{r,c}$, and the tolerance. For each station r, the *outputs* are the throughput $X_{r,c}$, the number of customers $N_{r,c}$, the response time $R_{r,c}$, and the station utilization $U_{r,c}$.
The approximate algorithms available are:
— `Chow` [Cho83]: assumes that the station queue lengths are identical in the model with full customer population and in the model with one less customer.

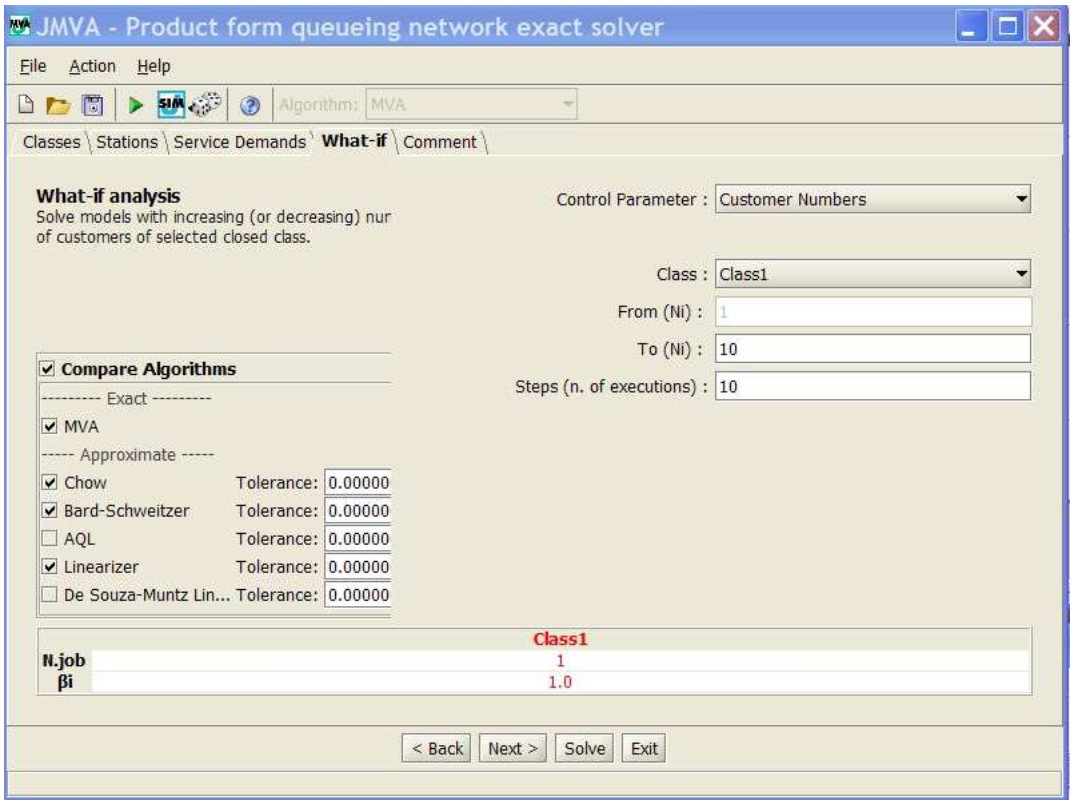


Figure 2.27: Checkboxes for the selection of the algorithms used in the `what-if` analysis

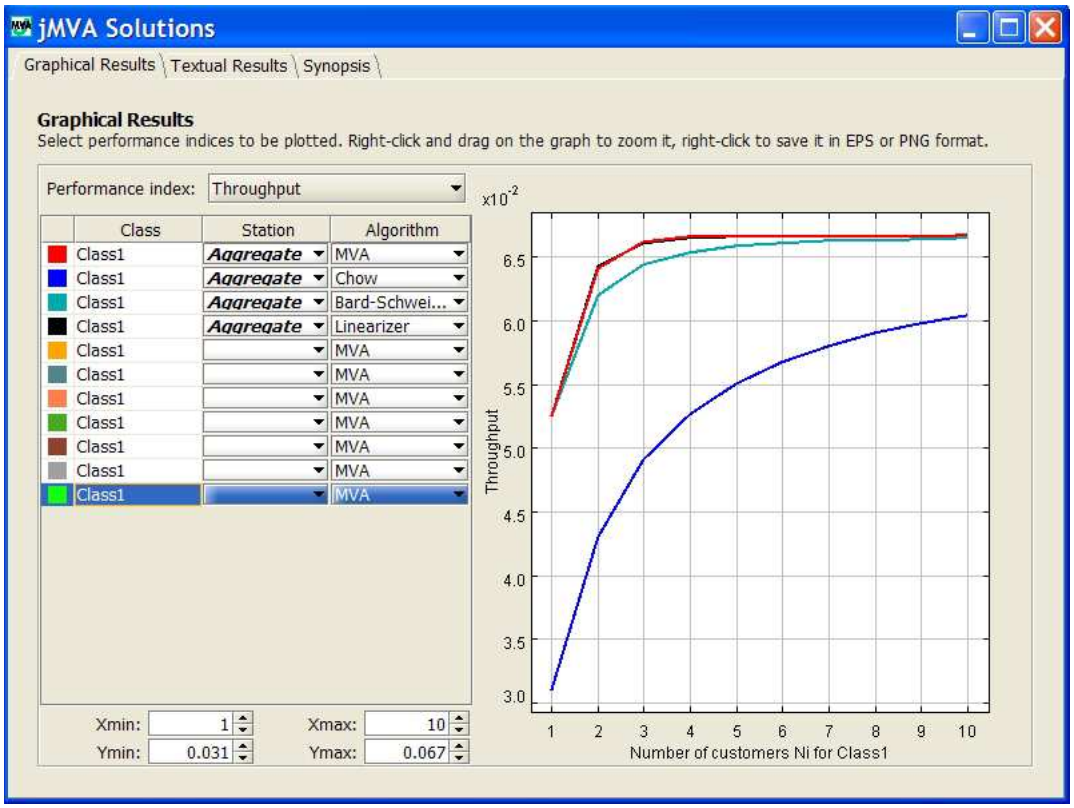


Figure 2.28: Graphics for the comparison of the performance indexes generated by the algorithms selected in the checkboxes of Fig.2.27.

— `Bard-Schweitzer` [Sch79]: assumes that removing a customer from a class affects the queue lengths of only that class and no others. It is more accurate than *Chow's* algorithm.
— `Linearizer` [CN82]: approximates queue lengths for populations with one less customer using information about full population. Using these estimates, new values are computed for full population, and the process is repeated.
— `De Souza-Muntz Linearizer` [dSeSM90]: reduces complexity of *Linearizer* by a factor of C (no. of classes). Instead of computing queue lengths for each class and station for N-1 populations, only computes them for each station, as only those are ever used.
— `AQL (Aggregated Queue Length)` [ZES88]: improves the *Linearizer* by a factor of C in terms of time and space, while maintaining almost same accuracy. It uses aggregate per-server queue lengths instead of per-class queue lengths as in *Linearizer*.

Specialized AMVA algorithms are also offered for processing models with priority, in particular `Chandy-Lakshmi` [CL83], `Bryant-Krzesinski-Teunissen` [BKT83], `Teunissen` [BKLC84], and Sevcik's *Shadow Server* method [oTCSRGS77].

2.4 Menu entries

2.4.1 File

New

Use this command in order to create a new JMVA model.
Shortcut on Toolbar: 
Accelerator Key: CTRL+N

Open

Use this command to open an existing model. You can only open one model at time, to edit two or more models start more than one instance of JMVA. If the current model was modified since its creation or last save action, a popup window will be shown for confirmation.
It is possible to open not only models saved with JMVA (*.jmva), but also with other programs of the suite (for example JABA *.jaba, JSIM *.jsim and JMODEL[3] *.jmodel). Whenever a data file of another tool is opened, a conversion is performed and error/warnings occurred during conversion will be reported in a window.
Models are stored in XML format, see *JMT system manual* for a detailed description.
Shortcut on Toolbar: 
Accelerator Key: CTRL+O

Save

Use this command in order to save the active document with its current name in the selected directory.
When you save a document for the first time, JMVA displays the Save As dialog box so you can rename your document. If you save a model after its resolution, results are stored with model definition data.
Shortcut on Toolbar: 
Accelerator Key: CTRL+S

Exit

Use this command in order to end a JMVA session. You can also use the Close command on the application Control menu. If current model was modified since its creation or last save action, a popup window will be shown for confirmation.

Accelerator Key: CTRL+Q

[3]In previous versions of the JMT suite, JMODEL was the name of the JSIMgraph application. Thus, *.jmodel files are files saved in previous versions of the tool.

2.4.2 Action

Solve

Use this command when model description is terminated and you want to start the solution of the model. At the end of the process the window in Figure 2.20 will popup.

Shortcut on Toolbar: 
Accelerator Key: CTRL+L

Randomize

Use this command in order to insert random values into Service Demands - or Service Times - table. Generated values are automatically adjusted to avoid saturation of resources.

Shortcut on Toolbar: 
Accelerator Key: CTRL+R

Import in JSIM

This command will import the current model into JSIM to solve it using simulator. A simple parallel topology is derived from the number of visits at each station and the generated model is equivalent to original one.

Shortcut on Toolbar: 

Accelerator Key: CTRL+G

2.4.3 Help

JMVA Help

Use this command to display application help. When you click on a line of the table of content you may access directly to the pages of the manual that describe the argument selected.

Shortcut on Toolbar: 
Accelerator Key: CTRL+H

About

Use it in order to display information about JMVA version and credits.

2.5 Examples

In this section we will describe some examples of model parametrization and solution using MVA exact solver. Step-by-step instructions are provided in five examples:

1. A single class closed model with three load independent stations and a delay service center (subsection 2.5.1)
2. A multiclass open model with two classes and three load independent stations (subsection 2.5.2)
3. A single class closed model with a load dependent station and a delay (subsection 2.5.3)
4. A multiclass mixed model with three stations (subsection 2.5.4)
5. A multiclass closed model where a what-if analysis is used to find optimal Population Mix values (subsection 2.5.5)

2.5.1 Example 1 - A model with a single closed class

Solve the single class model specified in Figure 2.29. The customer class, named *ClosedClass* has a population of $N = 3$ customers.

There are four stations, three are of load independent type (named *CPU*, *Disk1* and *Disk2*) and one is of delay type (named *Users*). *Users* delay station represents user's *think time* ($Z = 16$ s) between interaction with the system. Service times and visits for stations are reported in Table 2.3.

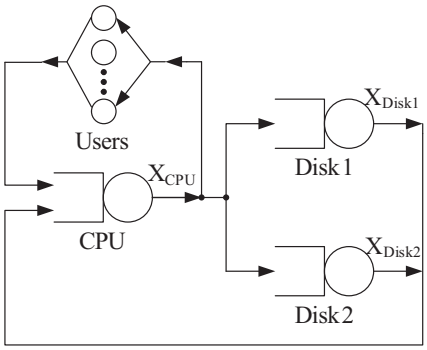


Figure 2.29: Example 1 - network topology

	CPU	Disk1	Disk2	Users
Service Times [s]	0.006	0.038	0.030	16.000
Visits	101.000	60.000	40.000	1.000

Table 2.3: Example 1 - service times and visits

Step 1 - Classes Tab

- use New command to create a new jMVA document
- by default, you have already a `Closed` class
- if you like, substitute default *Class1* name with a customized one (*ClosedClass* in our example)
- complete the table with workload intensity (number of customers). Remember that intensity of a closed class N must be a positive integer number; in this case, 3 customers

At the end of this step, the `Classes Tab` should look like in Figure 2.30.

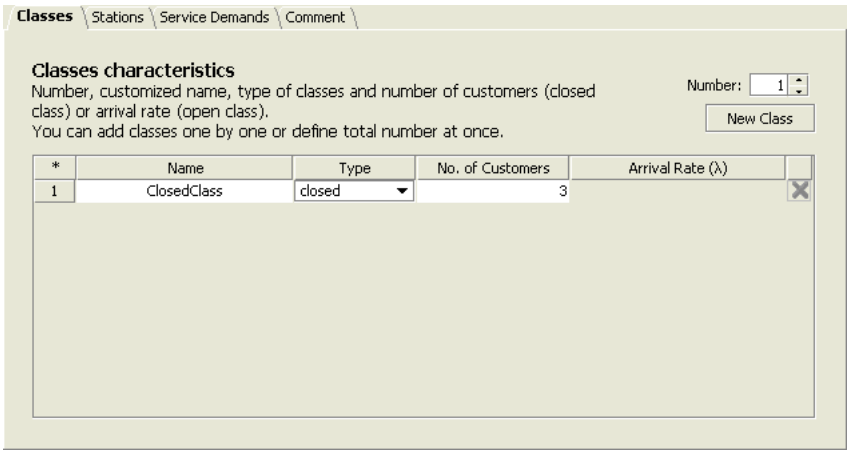


Figure 2.30: Example 1 - input data (Classes Tab)

Step 2 - Stations Tab

- click on `Next >` to switch to `Stations Tab`
- digit number 4 into stations number textbox or select number 4 using spin controls or push *New Station* button three times. Now your model has four `Load Independent` stations with a default name
- if you want you can change station names. Substitute *CPU* for default name *Station1*, substitute *Disk1* for default name *Station2*, substitute *Disk2* for default name *Station3* and substitute *Users* for default name *Station4*
- change the type of last inserted station; *Users* station is a `Delay (Infinite Server)`

At the end of this step, the `Stations Tab` should look like Figure 2.31.

Step 3 - Service Times and Visits Tabs

- use `Next >` command to switch to `Service Demands Tab`

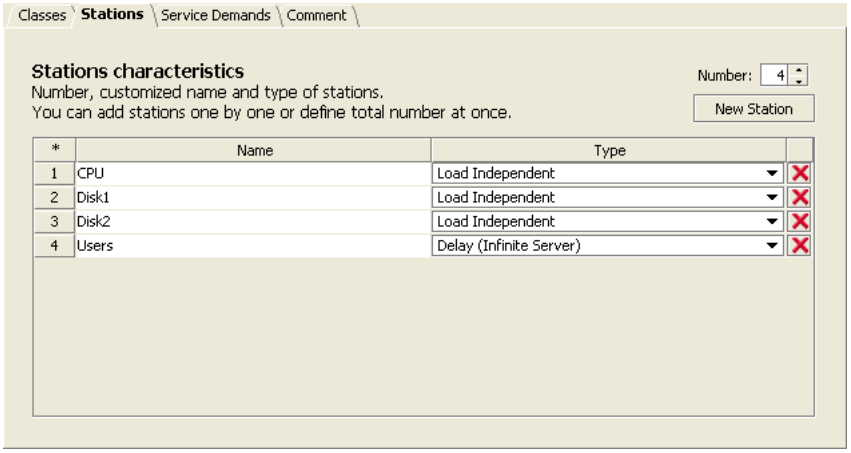


Figure 2.31: Example 1 - input data (Stations Tab)

- press *Service Time and Visit* button as you do not know the Service Demands of the three stations: in this case Service Times and number of Visits should be typed. After button pressure, the `Service Demands Tab` will be hidden and `Service times Tab` and `Visit Tab` will appear
- you can input all Service Times in the table. Remember that Service Time of the *Users* station, of delay type, is *think time* Z, in this case 16 s

At this point, the `Service Times Tab` should look like Figure 2.32.

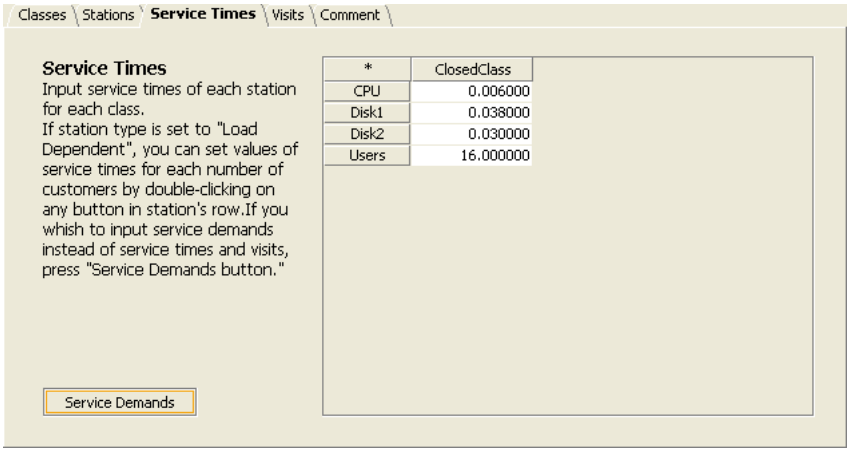


Figure 2.32: Example 1 - input data (Service Times Tab)

- use `Next` > command to switch to `Visits Tab`
- input numbers of visits for all centers in the table. In this case the number of visits of the *Users*, the infinite server station, is equal to 1 since a customer at the end of an interaction with the system visits this station.

At the end of this step, the `Visits Tab` looks like Figure 2.33.

Step 4 - Model Resolution

Use `Solve` command to start the solution of the input model. Model results will be displayed in a new window like the one of Figure 2.34.

Since we are considering a single-class model, all results in the column *Aggregate* correspond to the results in the *ClosedClass* column.

JMVA computes Residence Times W_k, Throughputs X_k, Queue lengths Q_k and Utilizations U_k for all stations k. The algorithm begins with the known solution for the network with zero customers, and progressively increases the population up to N that, in this example, is three. Note that the aggregate *Residence Time* is the *System Response Time* measure and the aggregate *Queue Length* (in the new versions this index has been referred to as *Number of Customers*) is the average number of customers in the system.

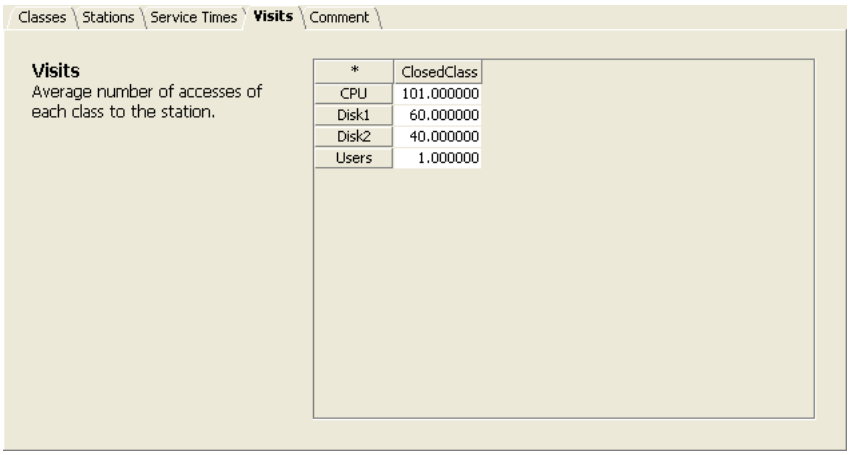


Figure 2.33: Example 1 - input data (Visits Tab)

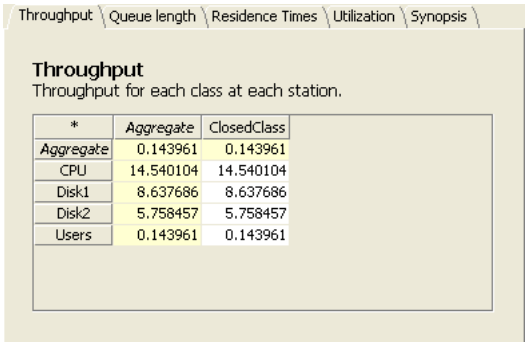


Figure 2.34: Example 1 - output data (Throughput Tab)

Using tab selector, you can change tab and see *Queue length, Residence Times, Utilizations* and a synopsis panel with a schematic report of the model (Figure 2.35).

The computed performance indices are shown in Table 2.4. Note that the *utilization* of a delay station (*users*) is the average number of customers in the station, therefore it may be greater than 1.

	Aggregate	CPU	Disk1	Disk2	Users
Throughput [job/s]	0.144	14.540	8.637	5.758	0.144
Queue Length [job]	3.000	0.193	0.410	0.194	2.303
Residence Time [s]	20.839	0.643	2.847	1.349	16.000
Utilization	-	0.087	0.328	0.172	2.303

Table 2.4: Example 1 - model outputs

2.5.2 Example 2 - A model with two open classes

Solve the multiclass open model specified in Figure 2.36. The model is characterized by two open classes A and B with arrival rate (the workload intensity λ) respectively of $\lambda_A = 0.15$ job/s and $\lambda_B = 0.32$ job/s. There are three stations of load independent type, identified with names *CPU*, *Disk1* and *Disk2*. Service times and visits for stations are shown in Table 2.5 and Table 2.6.

Since this model is similar to the network of Figure 2.29 solved in subsection 2.5.1, we will show how to easily create it from a saved copy of Example 1:

1. Open the saved instance of Example 1 model
2. Go to `Classes Tab`, change *ClosedClass* name to A, change its type to `Open` and set its arrival rate to $\lambda_A = 0.15$ job/s.
3. Click on *New Class* button, set name of new class to B, change its type to `Open` and set its arrival rate to $\lambda_B = 0.32$ job/s.
4. Go to `Stations Tab` and remove *Users* delay center.
5. Go to `Service Times Tab` and set service times for Class B according to Table 2.5.

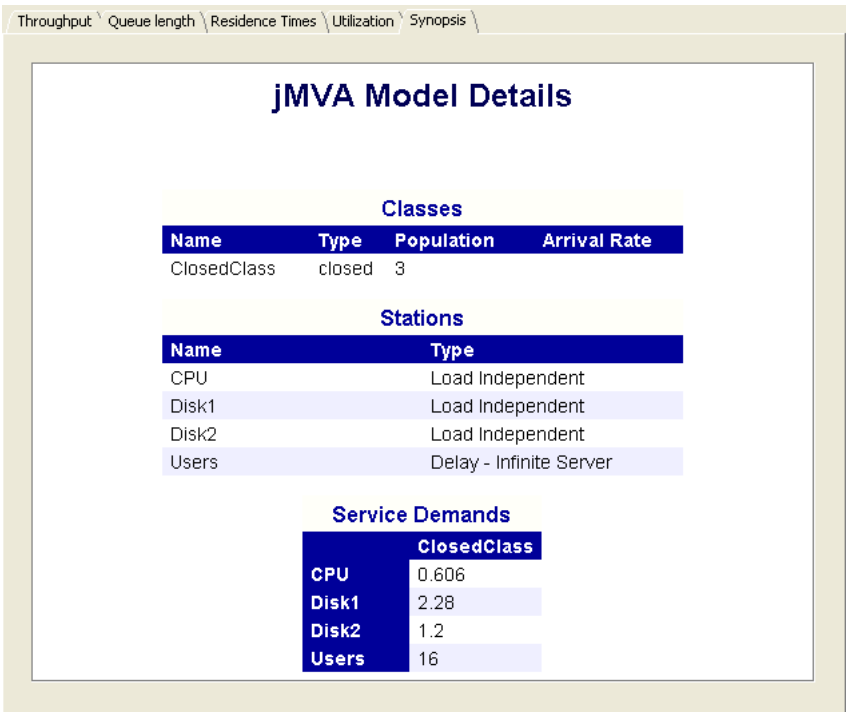


Figure 2.35: Example 1 - output data (Synopsis Tab)

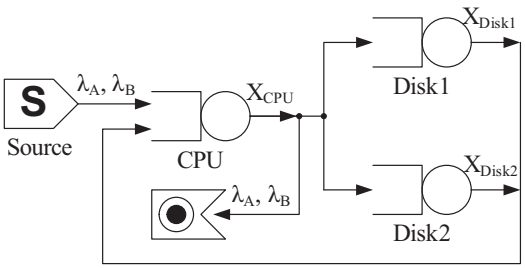


Figure 2.36: Example 2 - network topology

	CPU	Disk1	Disk2
Class A [s]	0.006	0.038	0.030
Class B [s]	0.014	0.062	0.080

Table 2.5: Example 2 - service times

	CPU	Disk1	Disk2
Class A	101.0	60.0	40.0
Class B	44.0	16.0	27.0

Table 2.6: Example 2 - number of visits

6. Go to `Visits Tab` and set visits for Class B according to Table 2.6.
7. Select `Solve` action.

The `Synopsis Tab` with a schematic report of the model created is shown on Figure 2.37, while the computed performance indices of this model are shown in Table 2.7.

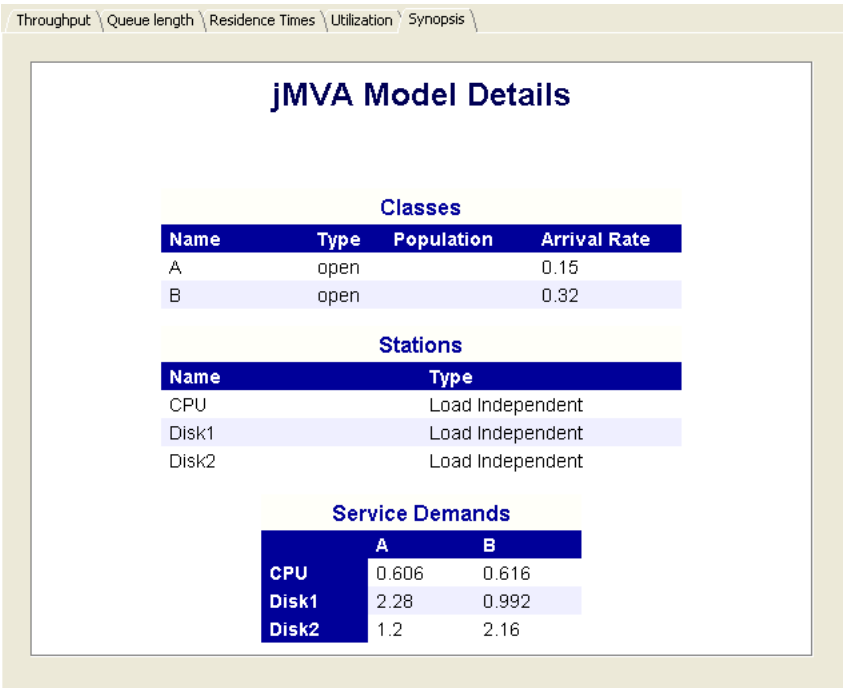


Figure 2.37: Example 2 - output data (Synopsis Tab)

	Class A			
	Aggregate	CPU	Disk1	Disk2
Throughput [job/s]	0.150	15.150	9.000	6.000
Queue Length [job]	2.529	0.128	1.004	1.398
Residence Time [s]	16.863	0.851	6.695	9.317
Utilization	-	0.091	0.342	0.180

	Class B			
	Aggregate	CPU	Disk1	Disk2
Throughput [job/s]	0.320	14.080	5.120	8.640
Queue Length [job]	6.575	0.277	0.932	5.366
Residence Time [s]	20.548	0.865	2.913	16.770
Utilization	-	0.197	0.317	0.691

Table 2.7: Example 2 - model outputs

2.5.3 Example 3 - A model with a load dependent station

The network is shown in Figure 2.38. It comprises only two stations: one is of delay type (named *Users*) and the other is a load dependent station (named *Station*). This model has one closed class only with $N = 8$ customers. The user's *think time* is $Z = 21$ s, while the service demands for the load dependent *Station*, shown in Table 2.8, are function of n: number of customers in the station ($D(n) = n + 1/n$).

n	1	2	3	4	5	6	7	8
$D(n)$ [s]	2.00	2.50	3.33	4.25	5.20	6.17	7.14	8.13

Table 2.8: Example 3 - service demands for *Station*, a load-dependent service center

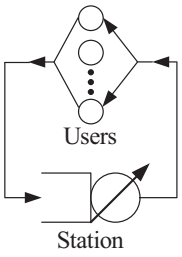


Figure 2.38: Example 3 - network topology

Step 1 - Classes Tab

The instructions that are the same given in Step 1 of subsection 2.5.1; in this case N must be 8.

Step 2 - Stations Tab

The instructions are given in Step 2 of subsection 2.5.1; in this case the model has two stations: a Load Dependent station and a Delay Center.

Step 3 - Service Demands Tab

1. use `Next` > command to switch to `Service Demands Tab`
2. double-click on cell with text *LD Setting...* to open the editor of load dependent service demands in a separate window, shown in Figure 2.39.
3. it is not mandatory to insert all values one-by-one. You can click or drag to select cells, enter the expression $n + 1/n$ into the textbox at the bottom of the window and click the *Evaluate* button.
4. at the end of this phase, editor window looks like Figure 2.40. Now you may press *OK* button to confirm changes and return to JMVA main window.

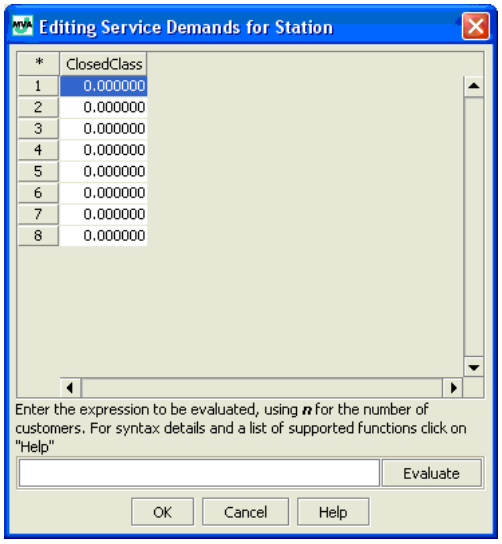


Figure 2.39: Example 3 - editor for the description of service demands for a load dependent station corresponding to the different number of customers before the parametrization

Step 4 - Model Solution

Use `Solve` command to resolve the model, results are shown in Table 2.10. A summary of the solution algorithms implemented in JMVA in given in Table 2.9 and are described later in the manual.

2.5.4 Example 4 - A model with one open and one closed class

The mixed queueing network model is shown in Figure 2.41. Workload intensities: the open class has an arrival rate $\lambda = 1$ job/s, the closed class has a customers number $N = 57$. Service demands are shown in Table 2.11.

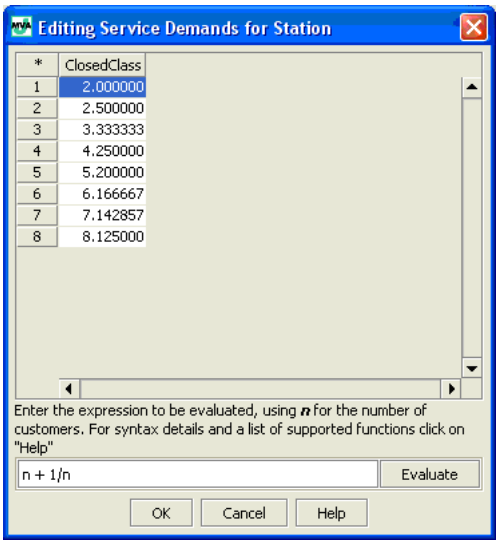


Figure 2.40: Example 3 - editor for the description of service demands for a load dependent station. In this case an arithmetic function has been defined: $S(n) = n + 1/n$

Table 2.9: Solution algorithms implemented in JMVA.

ALGORITHMS	
	EXACT
MVA	Mean-Value Analysis
RECAL	Recursion by Chain Algorithm
CoMoM	Class-Oriented Method of Moments
Tree MVA	Mean-Value Analysis for Sparse Networks
	APPROXIMATE
Chow	Approximate MVA
Bard-Schweitzer	Approximate MVA
AQL	Aggregate Queue Length Approximate MVA
Linearizer	Approximate MVA
DeSouza-Muntz	Faster Linearizer, Approximate MVA
Logistic Sampling	Normalizing Constant Approximation
	PRIORITY
Chandy-Lakshmi	Preemptive Priority Queues Approximate MVA
Bryant-Krzesinski-Teunissen	MVA for Preemptive and Non-Preemptive Priorities
Shadow Server	Sevcik's MVA Preemptive Resume Priority Approximation

Step 1 - Classes Tab

Follow the instructions of Step 1 in the previous examples; the `Classes Tab` is shown in Figure 2.42.

Step 2 - Stations Tab

Follow the instructions of Step 2 in the previous examples; in this case the model has three Load Independent stations (see Figure 2.43).

Step 3 - Service Demands Tab

Follow the instructions of Step 3 in the previous examples and define service demands for both classes as illustrated in Table 2.11 (see Figure 2.44).

	Aggregate	Station	Users
Throughput [job/s]	0.234	0.234	0.234
Queue Length [job]	8.000	3.080	4.920
Residence Time [s]	34.149	13.149	21.000
Utilization	-	0.810	0.973

Table 2.10: Example 4 - model outputs

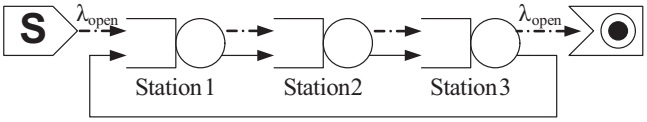


Figure 2.41: Example 4 - network topology

	Station1	Station2	Station3
OpenClass [s]	0.5	0.8	0.6
ClosedClass [s]	10.0	4.0	8.0

Table 2.11: Example 4 - service demands

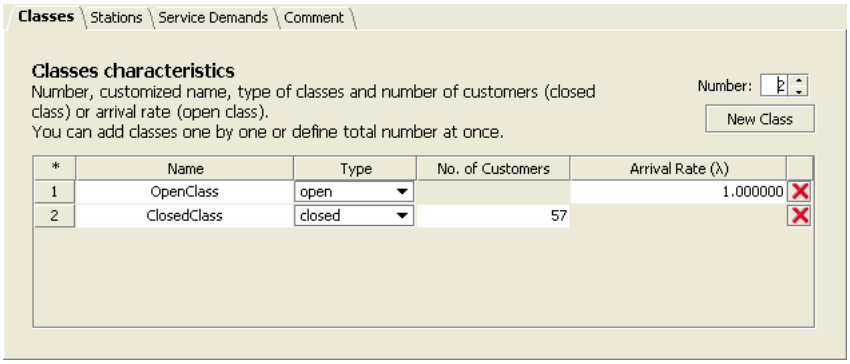


Figure 2.42: Example 4 - Class Tab

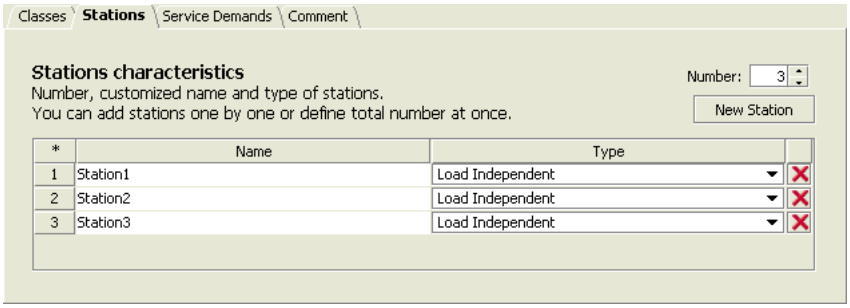


Figure 2.43: Example 4 - Stations Tab

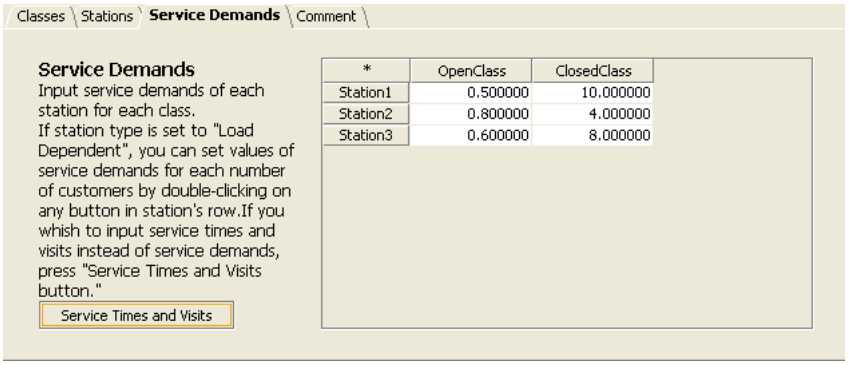


Figure 2.44: Example 4 - Service Demands Tab

Step 4 - Model Solution

Use `Solve` command. Results can be verified by computing the *equivalent model*, where the open class "slows down" the closed class by subtracting utilization to it:

$$\begin{aligned} D_1^{eq} &= \frac{D_{1,ClosedClass}}{1-\lambda * D_{1,OpenClass}} = 20s \\ D_2^{eq} &= \frac{D_{2,ClosedClass}}{1-\lambda * D_{2,OpenClass}} = 20s \\ D_3^{eq} &= \frac{D_{3,ClosedClass}}{1-\lambda * D_{3,OpenClass}} = 20s \end{aligned}$$

The MVA algorithm is used to solve the equivalent closed model. The number of customers of the closed class is 57, and the exact MVA technique should require the solution of other 56 models with smaller population. In this particular case, the formula used to compute the throughput can be simplified because the Service Demands are all identical:

$$\begin{aligned} X^{eq}(N) &= \frac{N}{\sum_{k=1}^{3} D_k^{eq} + \sum_{k=1}^{3} [D_k^{eq} * Q_k^{eq}(N-1)]} \\ &= \frac{N}{60 + 20 * \sum_{k=1}^{3} Q_k^{eq}(N-1)} \\ &= \frac{N}{60 + 20 * (N-1)} \end{aligned}$$

$$X^{eq}(57) = 0.048305 \text{ job/s}$$

The throughput for the closed class is $X_{ClosedClass} = X^{eq} = 0.048305$ job/s while the throughput for the open class coincide with its arrival rate $X_{OpenClass} = \lambda$. As visits were not specified, they have been considered equal to one: that is why throughput is equal at each station for each class (see Figure 2.45), i.e. the solved model consists of 3 stations that are sequentially connected with feedback.

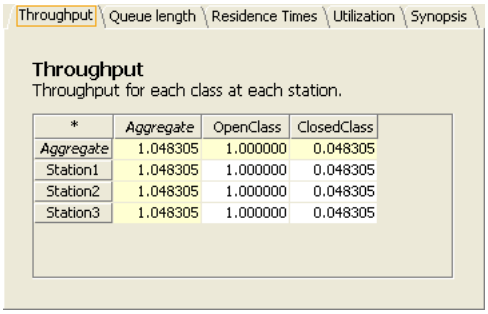


Figure 2.45: Example 4 - throughput

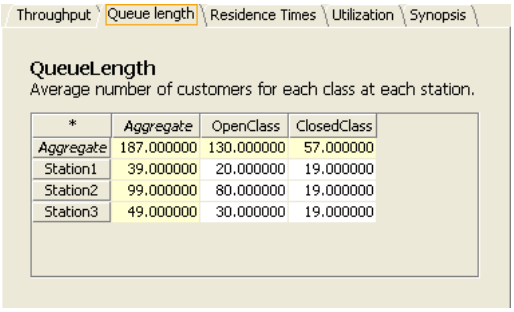
Queue lenghts can be computed with the following formulas:

$$\begin{aligned} Q_{k,ClosedClass}(N) &= Q_k^{eq}(N) \\ Q_{k,OpenClass}(N) &= \frac{\lambda * D_{k,OpenClass} * [1 + Q_{k,ClosedClass}(N)]}{1 - \lambda * D_{k,OpenClass}} \end{aligned}$$

And the results will be equal to the ones shown in Figure 2.46.

2.5.5 Example 5 - Identification of the Optimal Population Mix with a two-class workload

Let us perform a what-if analysis to find the optimal population mix that will maximize `System Throughput` and minimize `System Response Time` in the model shown in Figure 2.47. This point corresponds to the



*	Aggregate	OpenClass	ClosedClass
Aggregate	187.000000	130.000000	57.000000
Station1	39.000000	20.000000	19.000000
Station2	99.000000	80.000000	19.000000
Station3	49.000000	30.000000	19.000000

Figure 2.46: Example 4 - queue lengths



Figure 2.47: Example 5 - network topology

maximum value of `System Power` (see par.3.7). This model consists of two closed classes of customers *Class1* and *Class2*, and three load independent stations *Station1*, *Station2* and *Station3* with the service demands shown in Table 2.12. The total population of the model, i.e., the global number of customers of the two classes, is fixed to $N = 20$.

	Station1	Station2	Station3
Class1 [s]	1.0	5.0	1.0
Class2 [s]	5.0	1.0	5.0

Table 2.12: Example 5 - service demands

Step 1 - Classes Tab

Follow the instructions of Step 1 in the previous examples; as we will change population mix, initial allocation of the $N = 20$ jobs is irrelevant. For example we can allocate $N_1 = 10$ jobs to *Class1* and $N_2 = 10$ jobs to *Class2*. The `Classes Tab` is shown in Figure 2.48.

Step 2 - Stations Tab

Follow the instructions of Step 2 in the previous examples; in this case the model has three Load Independent stations (see Figure 2.49).

Step 3 - Service Demands Tab

Follow the instructions of Step 3 in the previous examples and define service demands for both classes as illustrated in Table 2.12 (see Figure 2.50).

Step 4 - What-if Tab

1. use `Next` > to switch to `What-if Tab`
2. Select `Population Mix` as a *control parameter* of the analysis in the combo box, several fields will be shown below.
3. *Class1* is already selected, by default, as reference class for the what-if analysis. This means that β_i values in `From` and `To` fields are referred to *Class1*.
4. By default, JMVA suggests the minimum allowed value of β_1 in the `From` field and its the maximum value in the `To` field[4]. Since we want to find the optimal value in the entire interval, we leave this unchanged.
5. we want to perform the maximum number of allowed executions, so we enter a big number in the `Steps` field (100 for example). JMVA will automatically calculate the maximum number of allowed executions provided that the number of customers for each class must be an integer and will report 19.

At the end of this phase, the What-if Tab will look like Figure 2.51.

[4]Since it is requested that one class has at least one job and customer number must be integer, the minimum value is $1/N$ and

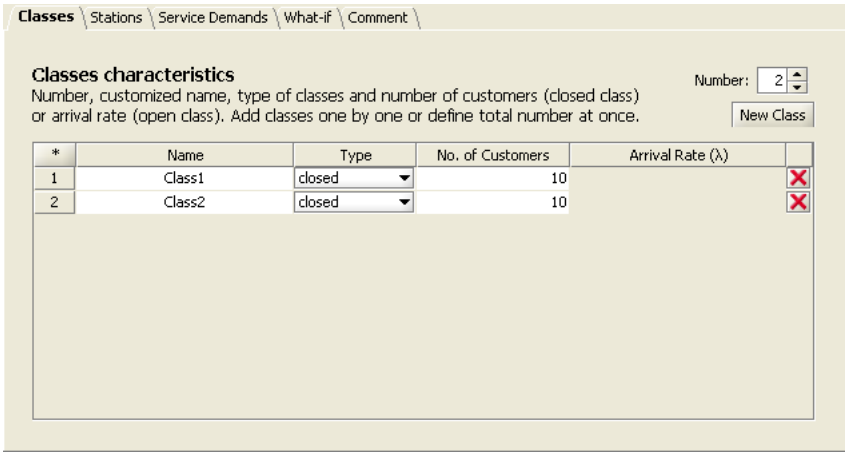


Figure 2.48: Example 5 - Class Tab

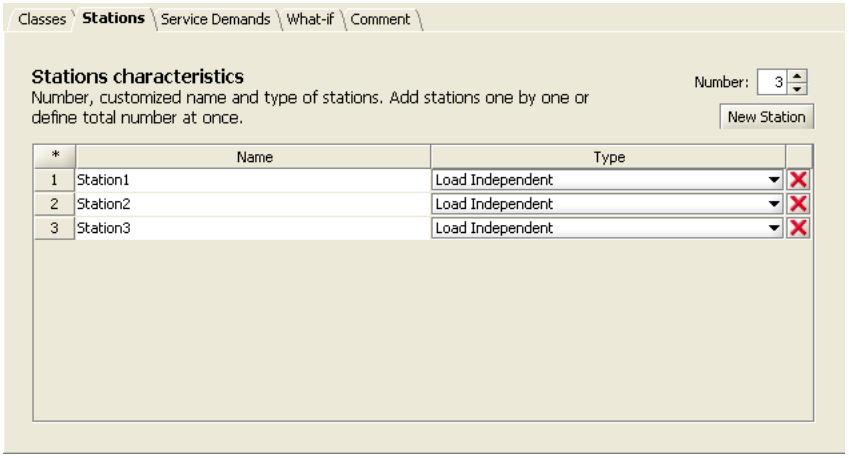


Figure 2.49: Example 5 - Stations Tab

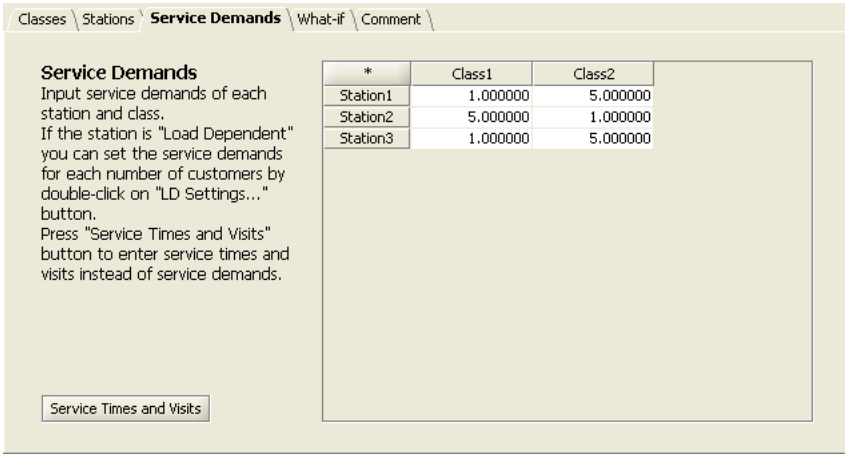


Figure 2.50: Example 5 - Service Demands Tab

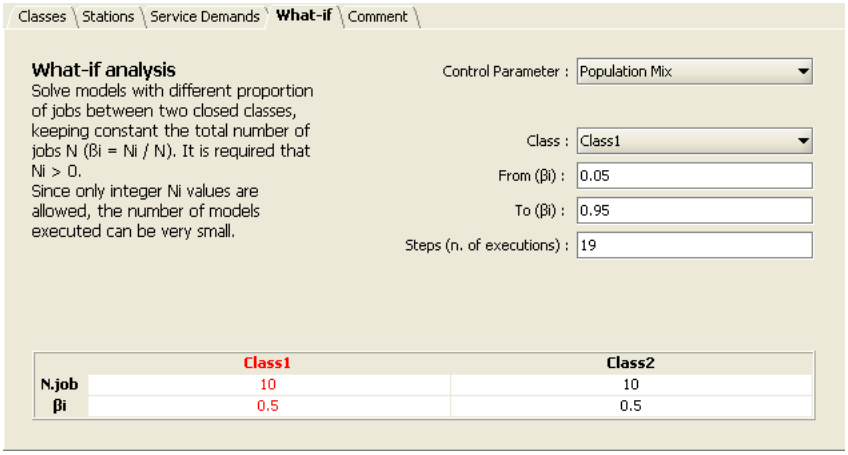


Figure 2.51: Example 5 - What-if Tab

Step 5 - Model Solution

Use `Solve` command. In the `Graphical Results Tab` select `System Response Time` and `System Throughput` as *Performance Index* (Figure 2.52 and Figure 2.53). Zooming on the plot, allows to identify the maximum value of System Throughput (0.32 job/s) and the minimum value of System Response Time (62.50 s). They both corresponds to the execution with population mix $\beta_1 = 0.40$ for *Class1*, which means that the optimal population values are $N_1 = 8$ and $N_2 = 12$.

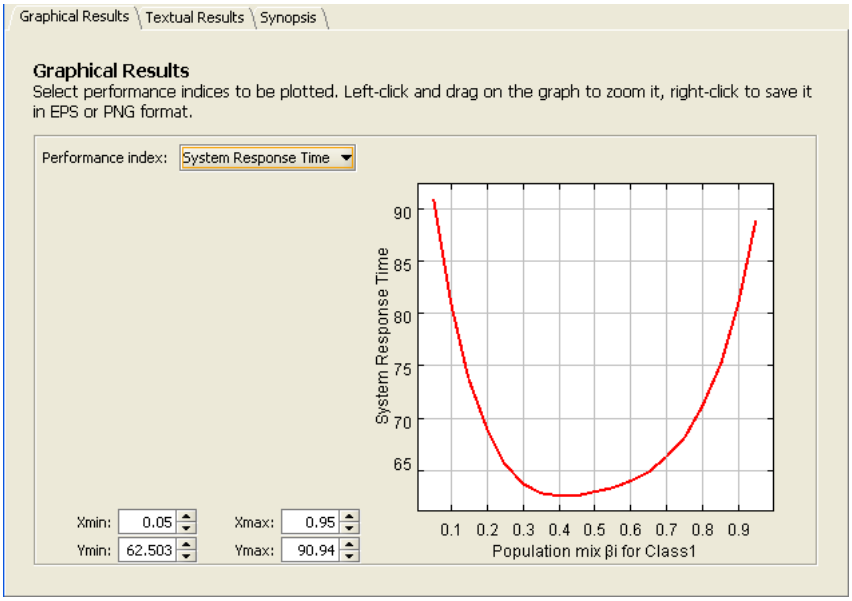


Figure 2.52: Example 5 - System Response Time

2.6 Command line for JMVA

JMVA has been designed to be very flexible. One important feature is the complete separation between GUI and computation engine obtained through an XML layer as shown in Figure 2.54. This architecture allows reuse of the analytic engine in other projects by simply providing a suitable XML input file (see subsection 2.6.1 for details). The model solution with analytical engine can be started with the *JMT.jar* file invoked as following:

```
java -cp JMT.JAR jmt.commandline.Jmt mva <File.xml>.
```

The *JMT.jar* file is located in the same directory where you installed the JMT tool.

The `File.xml` parameter is a well formed XML File according to the *JMTmodel.xsd* schema described in subsection 2.6.1. At the end of the computation, performance indices will be placed into the **solutions** element of a new file which has the same filename with the input file but with "-result.jmva" appended.

the maximum value is $(N-1)/N$.

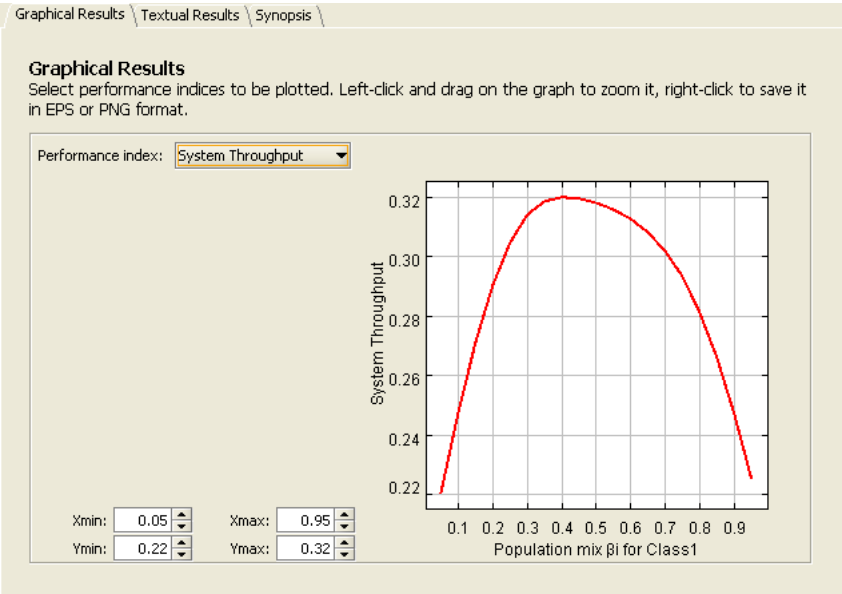


Figure 2.53: Example 5 - System Throughput

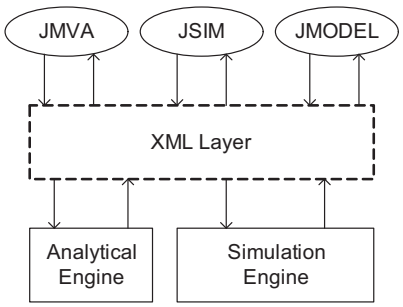


Figure 2.54: JMT Architecture

Moreover, running `java -cp JMT.JAR jmt.commandline.Jmt jmva <File.xml>`, it is possible to open the model directly in JMVA.

2.6.1 XML file format

JMVA XML format is simple and can be written even manually. The syntax of that file is specified in *JMT-Model.xsd* and is graphically represented in Figure 2.55.

The root element is **model** and can include an optional **description** of the model, a section with **solutions** (that is parameterized by the solver engine) and the section **parameters** used to specify the network to be solved.

The section **parameters** contains the section **classes** that has for attribute the global **number** of classes and can include from one to infinite **openclass** or **closedclass**. **openclass** must specify a **name** and arrival **rate**, while **closedclass** must specify a **name** and **population**.

For example to define one closed class with $N = 10$ and one open class with $\lambda = 0.5$ the code will be:

```
<classes number="2">
    <closedclass name="ClosedClass" population="10"/>
    <openclass name="OpenClass" rate="0.5"/>
</classes>
```

The other section of **parameters** is **stations**. This element has **number** as attribute (like classes) and is composed from one to infinite elements of the following types:

delaystation is a delay station (infinite server)

listation is a load independent station

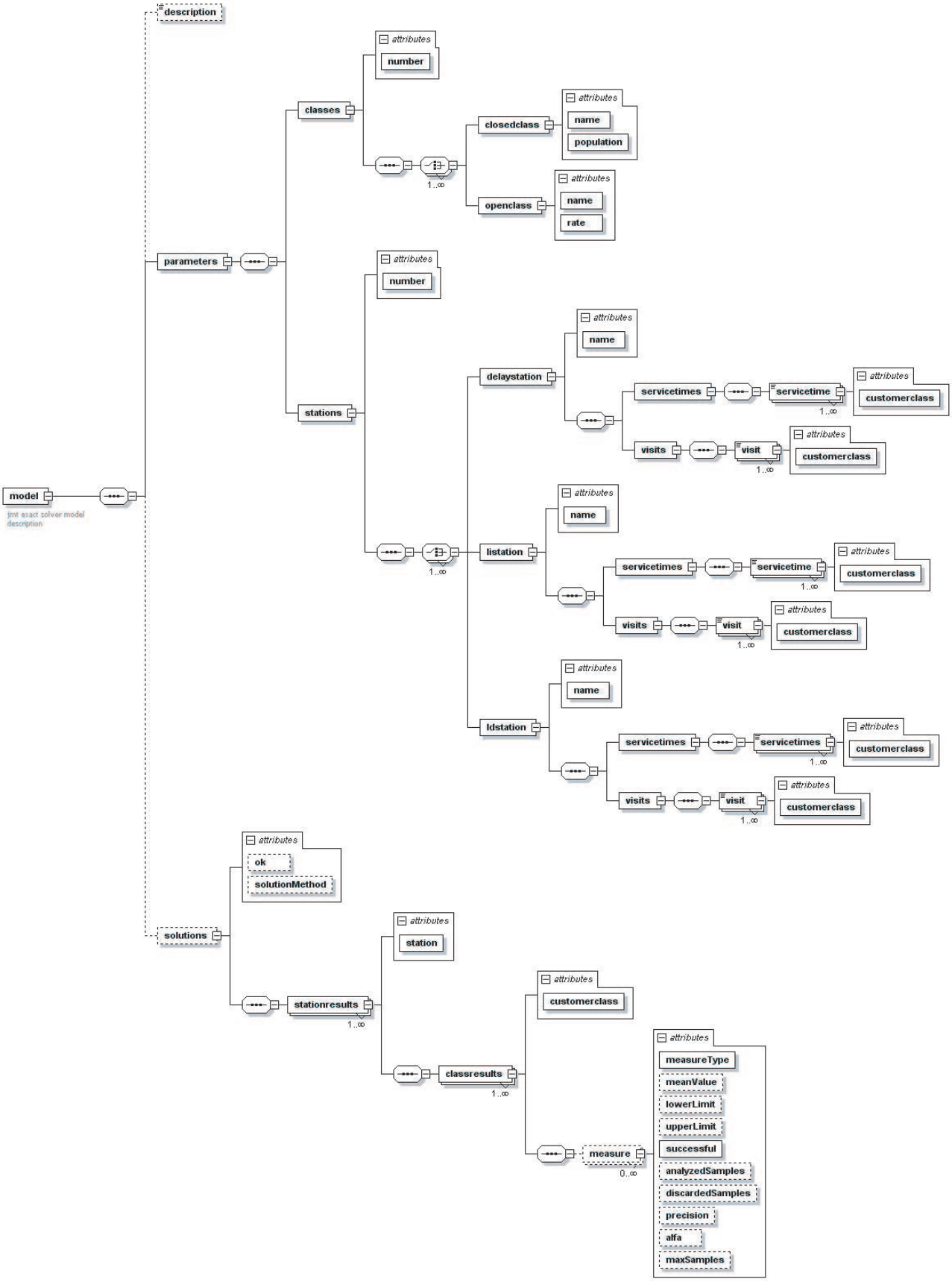


Figure 2.55: JMTModel.xsd stylesheet graphic representation

ldstation is a load dependent station

All the elements are defined with the same procedure. For each station the **name** attribute should be specified and two elements named **servicetimes** and **visits** must be included. **servicetimes** must include a list of at least one **servicetime** element and **visits** must include a list of at least one **visit** element. Both **visit** and **servicetime** elements have a `double` numeric value and each element has an attribute named **customerclass** that is used to associate a value of service time/visit with the corresponding customer class.

The only exception is **servicetime** parameter for a load dependent station. In this case the value is not a double but is a list of `double` separated by the special character `;` . The list is ordered for ascending values of number of jobs in the station, starting from 1 to number of customer N of the closed class of the model.

For example in a model with one closed class, named *Class1*, with $N = 5$ customers and three stations: a load independent with service demand 1, a delay with service demand 2 and a load dependent with $D(N) = N + 2$, the following code should be used for the station definition:

```
<stations number="3">
    <listation name="LoadIndependent">
        <servicetimes>
            <servicetime customerclass="Class1">1.0</servicetime>
        </servicetimes>
        <visits>
            <visit customerclass="Class1">1.0</visit>
        </visits>
    </listation>
    <delaystation name="Delay">
        <servicetimes>
            <servicetime customerclass="Class1">2.0</servicetime>
        </servicetimes>
        <visits>
            <visit customerclass="Class1">1.0</visit>
        </visits>
    </delaystation>
    <ldstation name="LoadDependent">
        <servicetimes>
            <servicetimes customerclass="Class1">3.0;4.0;5.0;6.0;7.0</servicetimes>
        </servicetimes>
        <visits>
            <visit customerclass="Class1">1.0</visit>
        </visits>
    </ldstation>
</stations>
```

Also the computed performance indices are stored in the same XML document. **solutions** has two attributes: **ok** that indicates if the computation was successful and **solutionMethod** that indicates if solution was obtained through analytical engine or simulator. Inside a **solutions** element there is a list of one or more **stationresults**, an element with station **name** as its attribute that contains one or more **classresults**, an element with class name (**customerclass**) as its attribute. This peculiar structure is needed to store a matrix of results. Each **classresults** element contains any number of **measure** elements that are used to store computed performance indices. **measure** has the following attributes:

measureType type of performance index, it can be one of te following *Queue length*, *Throughput*, *Residence time*, *Utilization*, and *Power*

successful boolean field that indicates if measure was computed correctly

meanValue (optional) computed value. This field is optional and will be always present if successful=true

lowerLimit (optional) lower limit of confidence interval. As MVA produces *exact* solution, this field is used only when model is solved with simulator

upperLimit (optional) upper limit of confidence interval. As MVA produces *exact* solution, this field is used only when model is solved with simulator

analyzedSamples (optional) number of samples analyzed by the simulator

discardedSamples (optional) number of samples discarded by the simulator

precision (optional) maximum relative error allowed by simulator

alfa (optional) confidence interval required to simulator

maxSamples (optional) maximum number of samples allowed by simulator

Note that only the first three elements of the above description will be saved by JMVA. The others are used when the model is solved using the simulator JSIM. For example, a model with one class (named *Class1*) and two stations (named *Station1* and *Station2*) will produce the following results:

```
<solutions ok="true" solutionMethod="analytical">
    <stationresults station="Station1">
        <classresults customerclass="Class1">
            <measure meanValue="9.930828575679609" measureType="Queue length" successful="true"/>
            <measure meanValue="5.467534818643117" measureType="Throughput" successful="true"/>
            <measure meanValue="1.8163265356477696" measureType="Residence time" successful="true"/>
            <measure meanValue="0.9999999999987986" measureType="Utilization" successful="true"/>
        </classresults>
    </stationresults>
    <stationresults station="Station2">
        <classresults customerclass="Class1">
            <measure meanValue="0.06917142432039082" measureType="Queue length" successful="true"/>
            <measure meanValue="5.467534818643117" measureType="Throughput" successful="true"/>
            <measure meanValue="0.01265130019557098" measureType="Residence time" successful="true"/>
            <measure meanValue="0.06469628980696993" measureType="Utilization" successful="true"/>
        </classresults>
    </stationresults>
</solutions>
```

It is important to point out that JMVA stores and loads model and results in *exactly* the same XML format used to dialogue with the analytical engine. The best practice to learn rapidly its syntax is to create a model using JMVA GUI, store it in any place and open it with a text editor.

Chapter 3

JSIMgraph (Simulation - Graphical interface)

3.1 Overview

In the JMT *suite* a discrete-event simulator for the analysis of queueing network models is provided. Two user interfaces are available: alphanumerical (JSIM*wiz*) and graphical (JSIM*graph*).

JSIM*graph* is the GUI front-end to JMT simulation engine. It helps the users to perform an evaluation study in two ways. Firstly, critical statistical decisions, such as transient detection and removal, variance estimation, and simulation length control, have been *completely automated*, thus freeing the users from taking decisions about parameters s/he may not be familiar with. The simulation is automatically stopped when all performance indexes can be estimated with the required accuracy. Secondly, a user-friendly graphical interface allows the user to describe, both the network layout and the input parameters. Furthermore, the graphical interface also provides support for the use of advanced features (several of them are for networks with very general characteristics, usually referred to as *non-product-form* networks) like fork and join of customers, blocking mechanisms, regions with capacity constraints on population, state-dependent routing strategies, user-defined general distributions, import and reuse of log data. A module for *What-If Analysis*, where a sequence of simulations is run for different values of control parameters, particularly useful in capacity planning, tuning and optimization studies, is also provided.

The simulation engine performs on-line the statistical analysis of measured performance indices, plots the collected values, discards the initial transient periods and computes the confidence intervals.
Network topologies implemented and solved using JSIM*graph* can be exported in vector (e.g., eps, pdf) or raster (e.g., jpg, png) image formats.

Main Features

Arrival rates for open classes of customers generated by Source stations and station **service times** (for any type of station in open and closed models) can be generated according to the following distributions: Burst (General), Burst (MAP: Markovian Arrival Process), Burst (MMPP2: Markov-Modulated Poisson Process of order 2), Coxian, Deterministic, Erlang, Exponential, Gamma, Hyperexponential, Lognormal, Normal, Pareto, Phase-Type, Replayer, Uniform, Weibull

The following **Queueing disciplines** are available:

- **Non-preemptive**: First Come First Served (FCFS), Last Come First Served (LCFS), Random (RAND), Shortest Job First (SJF), Longest Job First (LJF), Shortest Expected Processing Processing Time (SEPT), Longest Expected Processing Time (LEPT), *all with or without priority*;
- **Preemptive**: Processor Sharing (PS), Generalized Processor Sharing (GPS), Discriminatory Processor Sharing (DPS), First Come First Served Preemptive Resume (FCFS-PR), Last Come First Served Preemptive Resume (LCFS-PR), Shortest Remaining Process Time (SRPT).
- **Polling**: Exhaustive, Gated, K-Limited

The **drop rules** in case of limited capacity stations are: Drop, Blocking After Service (BAS), Waiting Queue, Retrial.

Routing of the customers in the network, i.e., the path followed by the requests among the resources, can be described either probabilistically or according to the following strategies: Fastest Service, Least Utilization, Load Dependent Routing, Random, Round Robin (with and without weights), Join the Shortest Queue, Shortest Response Time, Power of k choices (with and without memory), Disabled (the values of the control parameters are evaluated on the stations connected in output to the considered one). These strategies can be further combined among themselves through the use of a *routing station*.

Other notable features of the simulator are:

- Load-dependent service time strategies
- Fork-and-join stations to model parallelism
- Simulation of complex traffic patterns and service times (e.g., burst)
- Blocking regions (in which the number of customers is limited)
- What-if analysis (with multiple control parameters)
- Customization of default values
- Import/Export of the model from/to JMVA, the exact solver (when the required analytic assumptions are satisfied)
- Logging of the data flowing in any part of the model (*Logger* station)
- Graphical visualization of the evaluated performance indices together with their confidence intervals
- Automatic transient detection and removal
- Automatic stop of the simulation when all performance metrics can be estimated with the required accuracy.

JSIM*graph* has a modular Java-based architecture that allows the introduction of new Java classes in the simulation engine without any modification to the source codes of the other classes.

3.2 The home window

In the initial window of the JSIM*graph* (Figure 3.1), all the icons in the toolbar except the first two (`Create a new model` and `Open` a previously saved model) are inactive. Choosing `Create a new model` from the toolbar or `New` from the `File` menu bar activates the other icons in the toolbar. From Figure 3.1, we can see that the

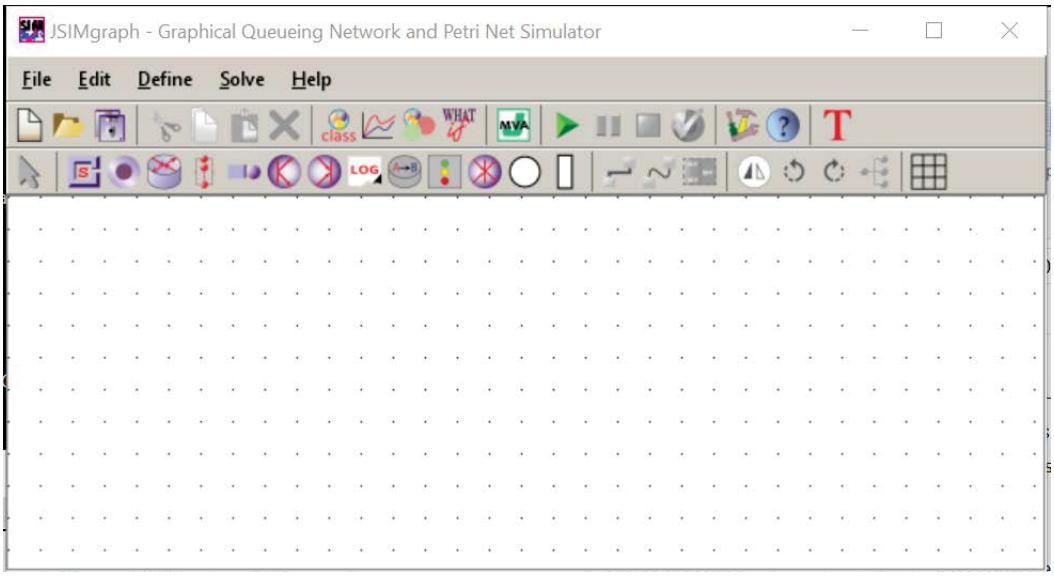


Figure 3.1: The home window of JSIM*graph*

home window has a *menu bar* and *two toolbars*. The menu bar has five menus: `File, Edit, Define, Solve, Help`.

The *first* toolbar has the following icons:

Create a new model

Open a saved model

 Save this model
 Cut
 Copy
 Paste
 Delete
 Define customer classes
 Define performance indices
 Define simulation parameters
 Define What-if analysis parameters
 Export current model to JMVA
 Start simulation
 Pause simulation
 Stop simulation
 Show simulation results
 Define default parameters
 Add/Use templates

The *second* toolbar has the following icons:

 Select
 Insert a new Source
 Insert a new Sink
 Insert a new Router
 Insert a new Delay
 Insert a new Queue
 Insert a new Fork
 Insert a new Join
 Insert a new Logger
 Insert a new ClassSwitch
 Insert a new Semaphore
 Insert a new Scaler
 Insert a new Place
 Insert a new Transition
 Connect two stations with a line
 Connect two stations with an arc
 Add selected stations to a new finite capacity region
 Rotate selected stations of 180 degrees
 Rotate selected stations connected only with arc connections of 45 degrees counterclockwise
 Rotate selected stations connected only with arc connections of 45 degrees clockwise
 Optimize the layout of a model whose stations are connected only with automatic connections (drawn with old graphical interface)
 Draw the grid

3.3 The graphical interface

With the release 1.1.0 of JMT, the graphical interface of JSIMgraph has been extended for better visualization of the multi-formalism models featuring both queueing and Petri Net elements. In the old graphical interface, JSIMgraph enables the fast drawing of simple models with several limitations, e.g., only straight lines are possible to connect two stations, the position of the connecting lines is automatic and the user cannot modify, several lines can be overlapped. We will refer to the connections drawn with it as *automatic connections* since

the user cannot change their shapes and positions.
Instead, in the new graphical interface the users may connect the stations with sharp or curved edges, referred to as *arcs*. The new can have different shapes and an arbitrary number of *intermediary points*, referred to as *control points*. Users can change the shape of the arcs in any run of the model. The shapes of the connections are implemented using *cubic Bezier curves*. Fig.3.2 shows an example of a model drawn with the new graphical interface.

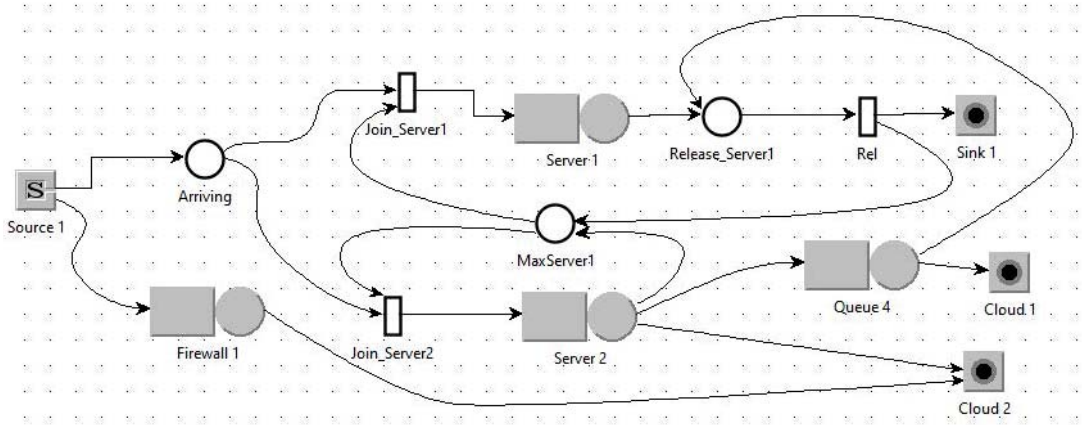


Figure 3.2: Example of a model implemented with the new graphical interface. The arcs are drawn using cubic Bezier curves.

CREATE AND EDIT ARCS: To create an arc between two stations, click the left button of the mouse on the first station and release it, then click on the second station or on several intermediate control points. Control points enable the user to draw arcs with any shape and edit them easily. When an arc is selected, with a click on the left button of the mouse, an `Editing tool box` will be displayed (Fig.3.3) with the possible operations on the arc.

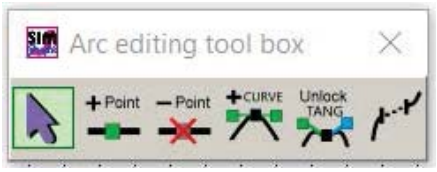


Figure 3.3: Tool Box for the editing of the arcs

The available operations on the arcs are:



Add a control point: select the arc on which you want to add a point, the editing box will be shown. Click on this icon, place the mouse over the arc (it will be highlighted in red), click the mouse and a control point will be added in the middle of the selected segment.



Delete a control point: select the arc on which you want to delete a point, the editing box will be shown, click on this icon, place the mouse over the point to be deleted (it will be highlighted in red), click the mouse and the control point is removed. Both the tangent points and the intermediate points can be deleted. When a tangent point is deleted, the tangent becomes null. When an intermediate point is deleted, the two arcs that originate and leave that point respectively are replaced by a single arc.



Add a tangent to a control point: select the arc with the control point to which you want to add a tangent, the editing box will be shown, click on this icon, place the mouse over that point (it will be highlighted in red), click the mouse and the tangent is added. The arcs arriving and departing a control point with tangent will follow a shape that is tangent to the segment drawn, that the user can modify as desired. Fig.3.4 shows the effects generated by two tangents on the shape of the arc that connect the stations `Queue1`

and `Sink2`.

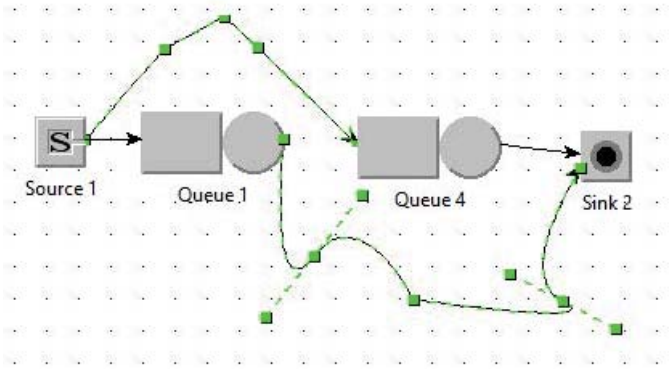


Figure 3.4: Example of the effects generated by the addition of tangents on two control points.



Breaking a tangent symmetry: by default the tangents are drawn as two segments symmetrical with respect to the control point of origin (the central point). Users can change the positions and length of the two segments of a tangent, with the effects of changing the sharpness of the curves. Click on this icon and select the control point with the tangent that you want to edit (it will be highlighted in red). Fig.3.5 shows some examples of the effects generated on the shape of the arcs by the asymmetry of tangents.

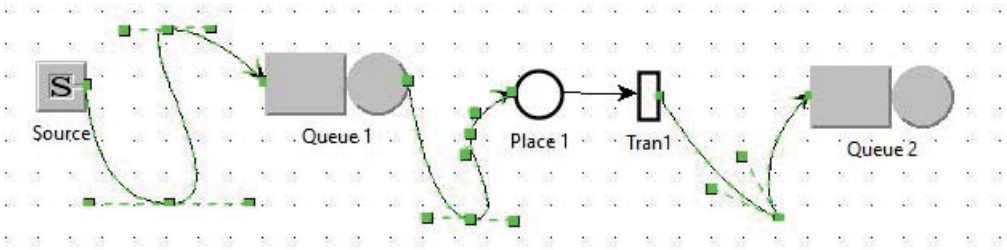


Figure 3.5: Example of the effects generated by the asymmetry of tangents.



Breaking a connection continuity: by default the connections are continuous lines. To break a connection: select the connection, select this icon in the toolbox, click on the control point where the arc will be broken (it will be highlighted in red). Once an arc has been broken, the two intermediate points obtained by the separation can be moved independently.

ROTATION OF STATIONS: Stations that have automatic, arcs or both types of connections can be rotated 180 degrees clicking on the icon . Stations that have only arcs connections can be rotated 45 degrees clockwise or counterclockwise clicking on the icons  and , respectively. In Fig.3.6 some examples of stations rotated 45 degrees clockwise an counterclockwise are shown.

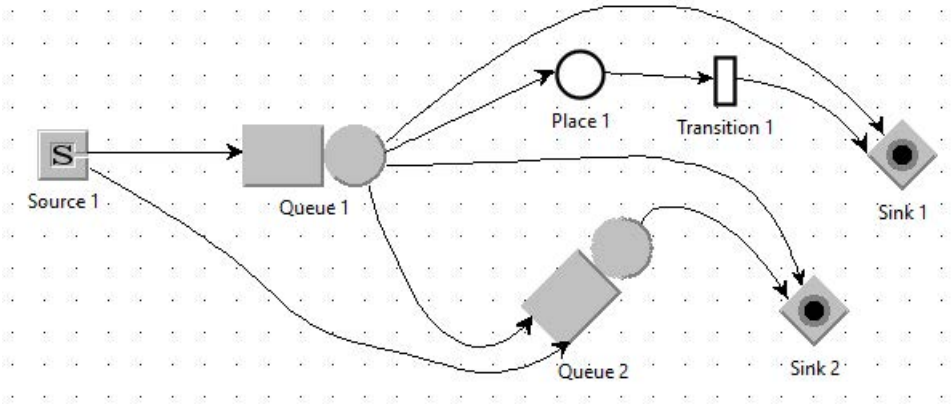


Figure 3.6: Examples of the rotation of stations with arc connections of 45 degrees clockwise an counterclockwise.

SWITCH BETWEEN THE TWO CONNECTION TYPES: The user can choose between connecting stations with non-modifiable automatic connections (drawn with the old interface style) or with Bezier arcs (drawn with the new interface style) which have free shapes and can be modified later. It is also possible to change the type of a connection of models loaded from the library from automatic to Bezier arc and from Bezier arc to automatic. Changing a connection type only affects the appearance of the connection and does not affects other model parameters.

COMPATIBILITY OF THE TWO GRAPHICAL EDITING STYLES: JMT supports two graphical editing style for use two different types of connection shapes: the automatic (the older one, straight lines non modifiable) and the arcs (the new one, based on Bezier curves that can be edited). The shapes of the connections implemented with Bezier arcs are saved in the XML file together with the other elements of the model. Complete compatibility is guaranteed for the models implemented with the two different editing styles. When a model is loaded from the library, the first step is to read its XML file. If a connection shape is found in the XML file, and if the model contains a corresponding connection for that connection shape, then the shape is added in the model in the form of a hash map that links the shape of the connection with the identifiers of the source and target stations. Then, when the arcs need to be rendered, the renderer checks whether a connection shape exists in the model, if yes, the connection is rendered according to the specified shape, otherwise the connection is rendered by dynamically computing a shape.
When a model created with the old editing style having all with automatic non-modifiable connections is opened, its XML file will not contain any connection shape. So, the connections will be rendered using the automatic shape computation of the old style (all straight lines). Therefore, there is no compatibility issue between the two versions of the graphical editing styles. After opening a model made with the older style, the user will have the possibility to change the connection types by transforming the automatic connections into Bezier arcs. If a model containing connections shapes is opened with old versions of JMT, the section of the XML file concerning the connection shapes will simply be ignored, so the connections will be displayed as if they were automatic connections.

3.4 Working with the graphical interfaces

3.4.1 Defining a new model

To define a new model the following steps have to be performed.
1. Draw the network (click and drop on the elements and the connections of tje old or the new graphical interface)
2. Define `Customer Classes` and select the `Reference Station` for each class
3. Set the parameters for each object
4. Select the performance indices to be collected and evaluated
5. If needed, insert one or more `Finite Capacity Regions` (FCR)
6. Choose or change the simulation parameters
7. Enable `What-If Analysis` and set its parameters, if required
8. Start the simulation
9. If diagnostic errors are detected, click on them and the related window that allows immediately to fix them

will be shown.

Step 1: Create the new model

To draw a new network, select or choose `New` from `File menu`. A white area in which the model should be drawn will be shown. Utilize the pre-defined objects like queueing station, delay, source, sink, logger, connection, routing station, place, transition and fork/join. Select the type of connections to be drawn. We suggest to utilize the new graphical interface to draw the arcs

Step 2: Define customer classes

In all the networks implemented, one or more *Customer Classes* must be defined. To describe them, see section 3.5 - "*Defining Customers Classes*" and parameterize the classes to use.

For each class of customers, a *Reference Station* should be set. This station is used to compute the *system throughput* for each class (i.e., the number of customers of that class that flow through the system in a time unit, and, in this case, that flow through the station selected as reference). It is a *required parameter* in order to compute correctly the simulation results. If a simulation is started without its prior definition, an error message will appear. Very often, the user station (usually a delay station) is selected as reference station. But this is not true on all the models. Thus, to provide the maximum of freedom to the users we ask this parameter as input for each model. Let us remark that the residence time of the reference station is accounted for the computation of the *system response time*.

Step 3: Define station parameters

All the stations have default parameters. The proper parameters of the objects should be defined: either the numeric values or the qualitative parameters such as routing strategy, queueing policy, maximum number of customers in an FCR, etc. See section 3.11 - "*Defining Network Topology*" for details.

Step 4: Define performance indices

The performance indices that will be evaluated during the simulation should be selected. Select the indices and set the class and the station which each index refers to. See section 3.7 - "*Performance Indices*" for details.

Step 5: Define a Finite Capacity Region

A *F*inite *C*apacity *R*egion (FCR) is a section of the model, that may consists of one or more stations, in which the number of customers is limited. Several FCRs can be defined in a network, provided that they *do not* overlap. See section 3.10 "*Finite Capacity Region (FCR)*" to learn how to set this type of region. **Step 6: Setting simulation parameters**

The parameters that control the simulation and the initial state of the network should be defined. See section 3.12 "*Modification of the parameters*" for details.

Step 7: Enabling a What-If Analysis

If you decide to set up a What-If Analysis read section 3.9.

Step 8: Start the simulation

Press or select `Simulate` from the `Solve` menu to start the simulation. Before starting the simulation, a check if the defined network is correct is performed, and the errors detected (fatal or warning) are shown. The results of a simulation are described in section 3.14.

Step 9: Fatal Errors and Warning Messages

The detected errors and warning messages are shown in section 3.13.

3.5 Classes of customers

The workload intensity is described by one of the following parameters:

- λ the arrival rate (for open models)
- N the population size (for closed models).

In *open* models the workload intensity is specified by the parameter λ, i.e., the rate at which requests (customers) arrive at the system. In these type of models there is an infinite stream of arriving customers and the customer population, i.e., the number of customers in execution, varies over time. Customers that have completed their execution leave the system (and reach a *sink station*). The class of customers of an open model is also referred to as *Open Class*.

In *closed* models the workload intensity is specified by a parameter N, indicating the average number of jobs (customers) in execution. In these models the population is fixed, i.e., N is kept constant. Customers that have completed their service are considered as if they left the model and have been replaced immediately by a new customer. The class of customers in execution in a closed model is also referred as *Closed Class*. Multiple class models consist of N customer classes, each of which has its own workload intensity and its own service

demand at each center. Within each class, the customers are indistinguishable, i.e., statistically equal. Models, in which the customers belong to both of the two types of classes, open and closed, are referred to as Mixed models. The parameters defining the two types of classes are given below.

- **Open Class parameters:** priority, interarrival time distribution, reference station, service time distributions of the stations
- **Closed Class parameters:** priority, population value, reference station, service time distributions of the stations.

The classes of customers identify different customer behavior and characteristics, such as the type (closed or open), the size of the customer population (for closed classes) or the interarrival time distribution (for open classes), the path among the resources, the service time required.

They can be set from the `Define` menu by choosing `Customer Classes`. The following panel will be shown.

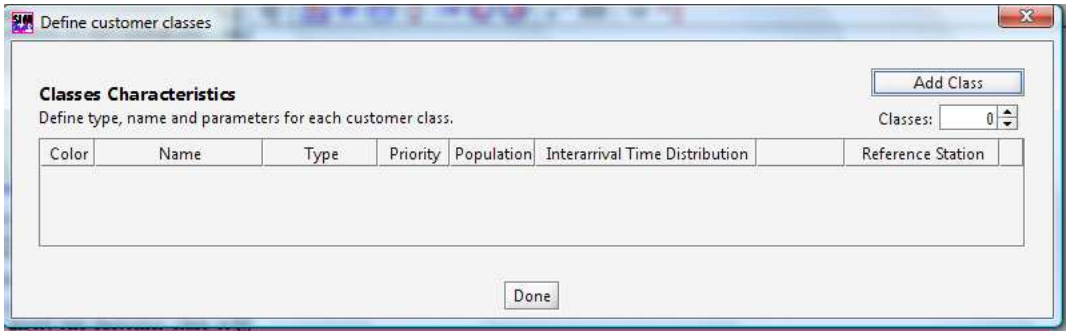


Figure 3.7: Window for the definition of the classes of customers

Classes must be explicitly added to the model, either one at a time by clicking the `Add Class` button, or by selecting directly the desired final number of classes from the `Classes` counter. The newly added classes will be listed with default parameters.

Double click on the default name (i.e., Class0, Class1, etc.) to change it. Each new class has a priority in the system. A smaller number indicates a lower priority. Default value is 0, it can be changed by double clicking on the corresponding area. In this panel you can insert either one or multiple customer classes.

Defining Open Classes

After adding a class and set its name and priority, you must select the type of the class. Classes are created *Closed by default*, so if you want an *Open* class, select the type *Open* from the `Type` menu.

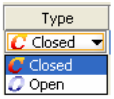


Figure 3.8: Selection of the type of the class

Now the class characteristics should look like Figure 3.9.

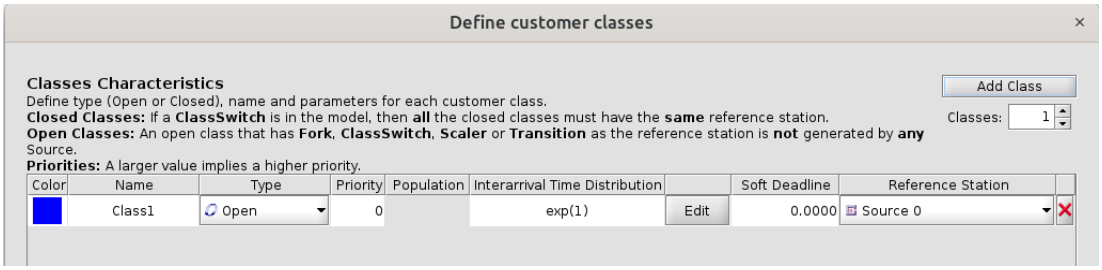


Figure 3.9: Class characteristics window

Open classes describe customer populations that vary over time. They are characterized by the probability distribution of the interarrival time of customers arriving at the system. The *default* Interarrival Time Distribution is *exp(1)* (Exponential Distribution with $\lambda = 1$). To change the Interarrival Time Distribution, click the `Edit` button.

The window of Figure 3.11 will appear. Click on the `Selected Distribution` drop-down menu to choose

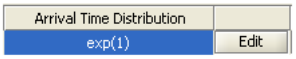


Figure 3.10: Definition of the interarrival time distribution and its parameters (`Edit` button)

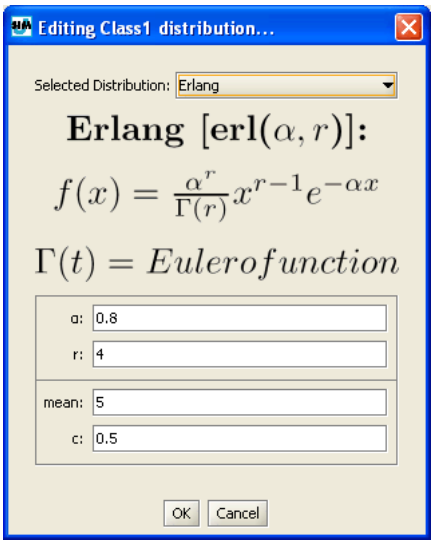


Figure 3.11: Window for the editing of the distribution of interarrival times

one of the distributions listed in Table 3.1.

For each distribution the correct values of the parameters should be described, *default* values can eventually be used. Parameters that are related each others are automatically updated when one of them is modified. For example, for an *Erlang* distribution, when you set the (α, r) pair, the $(mean, c)$ pair is automatically set to the correct values. The *Replayer* distribution allows the user to use trace of data collected from real experiments. Click `OK` to return to `Class parameters` definition.

The final key step of a class definition is the identification of the *Reference Station*, i.e., the station of the model used to compute the system throughput and response time of a class. It can be selected from the `Reference Station` menu (Figure 3.12). For *Open Classes*, *only* one of the stations of `Source` type can be selected as *Reference Station*.

The *Soft deadlines* value represents a target for the jobs of that class in terms of their total response time from entering the system to completing execution. If they miss the soft deadline, they continue executing until completion. Metrics such as system earliness, system tardiness, and system lateness can be used to study how timely the jobs have been compared to their deadlines.

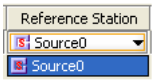


Figure 3.12: Selection of the Reference Station for an open class of customers

Defining Closed Classes

By *default* Classes are created as *Closed*. Priority can be changed as in the case of Open class (the default value is 0, the minimum). The population size (also referred to as N) is the parameter that characterizes a closed class. For a closed class N is constant and does not change during the execution of the simulation. Its *default* value is 1 and it may be changed by clicking on the corresponding area in the class properties matrix. A Reference Station for the class *must* be selected from the `Reference Station` menu. With a closed class all the type of stations can be selected but neither a *Source* nor a *Sink*. The *Reference Station* is used to compute the system throughput and response time for the considered class.

3.6 Distributions

Open class models are workloads where the number of requests in the system fluctuates over time and new requests arrive at the system as if generated by an infinite source. The arrival pattern can be described by a probability distribution simple or very complex. It is often used the distribution of the *inter-arrival* times of consecutive requests, or customers.

Table 3.1: Distributions available for *Interarrival* and *Service times*.

Stations	*Distributions*	
	`Acronym`	`Description`
`Source` `Queue`	Burst (general)	Two different distributions can be intermixed
	Burst (MAP)	Markovian Arrival Processes
	Burst (MMPP2)	Markov chain that jump between two states
	Coxian	Generalization of a hypoexponential distribution
	Deterministic	Deterministic sequence of values every k time units
	Erlang	Special case of a Gamma distribution with integer shape parameter
	Exponential	Distance between two consecutive events generated by a Poisson process
	Gamma	Distribution with two parameters: shape α, and scale λ
	Hyperexponential	coefficient of variation >1
	Lognormal	Product of independent positive random variables
	Normal	a Gaussian function, two parameters: mean μ and standard deviation σ
	Pareto	with two parameters: location k and shape α
	Phase-Type	Sequential connection of several Poisson processes
	Replayer	Trace of data provided by the user
	Uniform	May assume all the values in a range with constant probability
	Weibull	the rate of events increases or decreases over time as a function of k

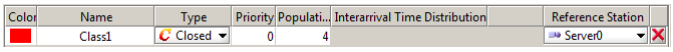


Figure 3.13: Closed class definition parameters

A probability distribution $f(x)$ can be characterized by

- *Mean* (if it exists):

$$E[X] = \int_{-\infty}^{\infty} x\, f(x)\, dx$$

- *Variance* (if it exists):

$$Var[X] = \int_{-\infty}^{\infty} (x - E[X])^2 f(x)\, dx = E[X^2] - E[X]^2$$

- *Coefficient of variation* c (when mean and variance exist and mean is not 0):

$$c = \frac{\sqrt{Var[X]}}{E[X]}$$

The following probability distributions are currently supported.

***Burst (General)* distribution**
The Burst (General) distribution is a complex distribution obtained combining two different distributions. It allows the simulation of complex patterns of arrivals and may be suitable for simulating the load generated by web 2.0 applications. A Burst (General) distribution consists of two different types of intervals (A and B) independent among each other. For both the interval types the following parameters have to be specified:

- the *probability* of *occurrence* of that type of interval in the generated stream
- the *length distribution* for that type of intervals
- the distribution of the *values* generated into each interval.

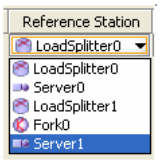


Figure 3.14: Selection of a Reference Station for a closed class of customers

The Burst (General) distribution behaves as follows. As soon as the simulation starts, an interval is chosen based on the specified probability of occurrence. If, for example, interval type A has probability of occurrence of 0.7 and interval B has 0.3, then an interval of type A is chosen with 70% probability. In order to determine the duration of an interval, an event from the interval-length distribution is obtained. The distribution of the events of an interval is then used to compute the arrival time of customers or the service time of a station, depending on where the Burst (General) distribution is used. As soon as the interval ends, a new interval is chosen based on the given probability as before.
Figure 3.15 shows the concept idea of the Burst (General) distribution, clarifying the role of the occurrence probability and the role of the interval length.

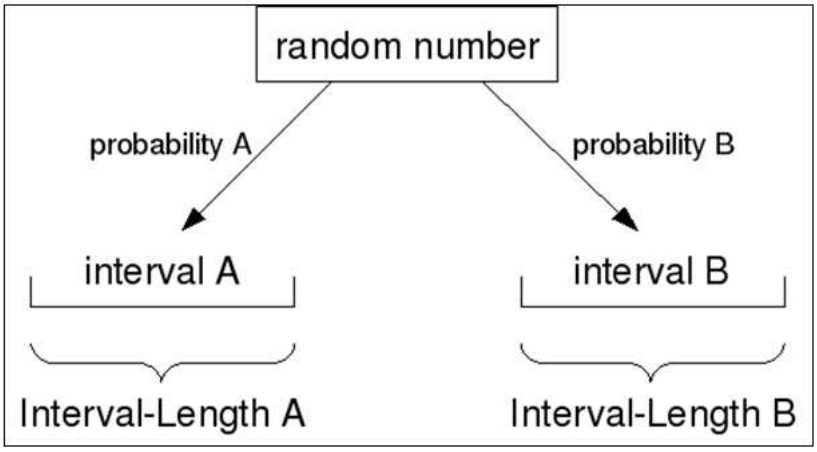


Figure 3.15: Main structure of the *Burst (General)* distribution consisting of two type of intervals with independent statistical characteristics

***Burst (MAP)* (map($\boldsymbol{D}_0, \boldsymbol{D}_1$)) distribution**
Markovian Arrival Processes (MAPs) [Neu89] are a general class of point processes useful for analyzing the impact of non-Poisson workloads in performance evaluation models.
A MAP(n) builds on a continuous-time Markov chain (CTMC) with n states, and can be specified by two $n \times n$ matrices, a stable matrix $\boldsymbol{D}_0$ and a non-negative matrix $\boldsymbol{D}_1$, that describe transition rates between the n states. Being observable, each transition in $\boldsymbol{D}_1$ produces a job arrival; $\boldsymbol{D}_0$ instead contains hidden transitions, which are not associated with arrivals. $\boldsymbol{Q} = \boldsymbol{D}_0 + \boldsymbol{D}_1$ is the infinitesimal generator matrix of the underlying CTMC. For a MAP(n), inter-arrival time moments and autocorrelations are computed using the probability vector $\boldsymbol{\pi}_e$ of the embedded process with irreducible stochastic matrix $\boldsymbol{P} = (-\boldsymbol{D}_0)^{-1}\boldsymbol{D}_1$, where $\boldsymbol{\pi}_e\boldsymbol{e} = 1$ and $\boldsymbol{e}$ is an $n \times 1$ vector of all ones. The MAP inter-arrival times are phase-type distributed with k-th moment

$$E[X^k] = k!\boldsymbol{\pi}_e(-\boldsymbol{D}_0)^{-k}\boldsymbol{e}$$

and squared coefficient of variation

$$c^2 = 2E[X]^{-2}\boldsymbol{\pi}_e(-\boldsymbol{D}_0)^{-2}\boldsymbol{e} - 1$$

Note that MAPs may be used to characterize service times as well. In this sense, each observable transition in $\boldsymbol{D}_1$ represents the service completion of a job. Additionally, Phase-Type distributions and Markov-Modulated Poisson Processes can be seen as special cases of MAPs.

***Burst (MMPP2)* (mmpp2($\sigma_0, \sigma_1, \lambda_0, \lambda_1$)) distribution**
A MMPP(2) is a continuous-time Markov chain that jumps between two states and the active state determines the current rate of service. For example, one state may be associated with slow inter-arrival times (state 0), the other may have fast inter-arrival times (state 1). As time passes, the MMPP(2) jumps several times between the slow state 0 and the fast state 1. In JMT, the MMPP(2) is parameterized by four rates: σ_0, σ_1, λ_0, and λ_1 (see Figure 3.16). While in state 0, the MMPP(2) generates arrivals with rate λ_0 or jumps to state 1 with rate σ_0; in state 1 the arrival and jump rates are λ_1 and σ_1, respectively. This can be equivalently stated as follows: in state 0 a random number with exponential distribution having rate $\lambda_0 + \sigma_0$ is drawn, then with probability $\lambda_0/(\lambda_0 + \sigma_0)$ is the arrival of a new job (or equivalently the completion of its service if the MMPP(2) is used as a service process), while with probability $\sigma_0/(\lambda_0 + \sigma_0)$ *without* any job arrival or service completion; the case of state 1 is similar.
A MMPP(2) can be parameterized to have a predefined mean, c^2, skewness, decay rate of the autocorrelation.

Given these four parameters, there are closed-form formulas to obtain the corresponding values of σ_0, σ_1, λ_0, and λ_1. For example the mean is related to the rates of the MMPP(2) by

$$mean = \frac{\sigma_0 + \sigma_1}{\lambda_0\sigma_1 + \sigma_0\lambda_1}$$

while the decay rate of the autocorrelation that controls the burstiness is

$$decay\ rate = \frac{\lambda_0\lambda_1}{\lambda_0\lambda_1 + \lambda_0\sigma_1 + \sigma_0\lambda_1}.$$

In particular, a large decay rate (> 0.9) creates large bursts of arrivals/service completions. The reader could find additional details on how to fit a MMPP(2) for example in [CZS07] and references therein.

- **Example 1)** $\lambda_0 = \lambda_1 = 5$ is an exponential distribution for all choices of σ_0 and σ_1.
- **Example 2)** $\lambda_0 = 1.4346$, $\lambda_1 = 0$, $\sigma_0 = 0.0139$, $\sigma_1 = 0.0319$ is an hyperexponential with mean= 1, $c^2 = 20$, $p = 0.99$. since the hyperexponential has no burstiness, here the autocorrelation coefficients are equal to zero.
- **Example 3)** $\lambda_0 = 12$, $\lambda_1 = 0.0879$, $\sigma_0 = 0.0010$, $\sigma_1 = 0.0001$ is a MMPP(2) with burstiness. The mean is 1, c^2=20, skewness= 7.31, lag-1 autocorrelation coefficient is 0.4745. Since for a MMPP(2) the autocorrelation is always less than 0.5, this is an example of a process with very large burstiness.

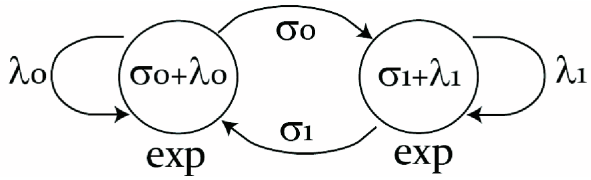


Figure 3.16: Main structure of the *Burst (MMPP2)* distribution

Coxian **(cox($\lambda_0, \lambda_1, p_0$)) distribution**
The *Coxian* distribution is a generalization of the hypoexponential distribution, which can be used to model empirical distributions with a coefficient of variation lower than, identical to or greater than 1. This distribution is characterized by two rate parameters λ_0 and λ_0 as well as a probability parameter p_0 (see Figure 3.17) such that

- with probability p_0, it acts as an exponential distribution with rate λ_0
- with probability $1 - p_0$, it is the convolution of two exponential distributions with rates λ_0 and λ_1, respectively.

In the case of $\lambda_0 = \lambda_1$ and $p_0 = 0$, the Coxian distribution reduces to an Erlang distribution with shape $r = 2$ and rate $\alpha = \lambda_0 = \lambda_1$. If the *mean* and the *coefficient of variation* c are set, the value of c^2 must be no less than 0.5.

Deterministic **(det(k)) distribution**
This distribution describes a deterministic (constant) flow of customers, arriving exactly every k time units. The probability density function is:

$$f(x) = \begin{cases} 1 & x = k \\ 0 & x \neq k \end{cases}$$

The only parameter that should be specified is the *mean*, equal to k.

Erlang **(erl(α, r)) distribution**
The *Erlang* distribution is a continuous distribution, with two parameters: the *shape* r, an integer value, and the *rate* α, a real value. It is a special case of a Gamma distribution with integer shape parameter. The probability density function is:

$$f(x) = \frac{\alpha^r}{\Gamma(r)}\, x^{r-1}\, e^{-\alpha x}$$

$\Gamma(r)$ is called Eulero function and when r is a non null positive integer, as in the case of the Erlang distribution, the Eulero function reduces to $\Gamma(r) = (r-1)!$.

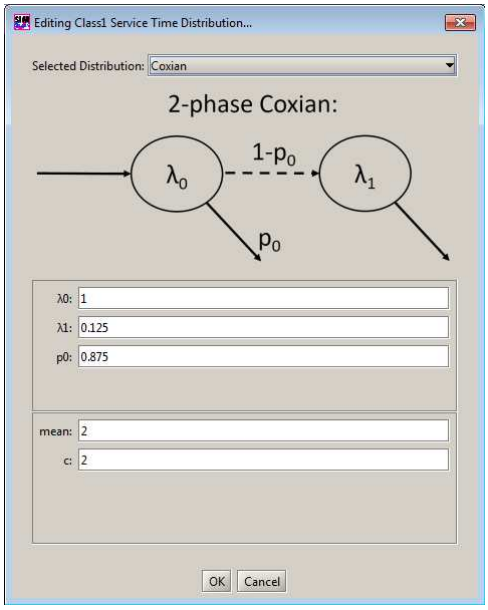


Figure 3.17: Main structure of the Coxian distribution

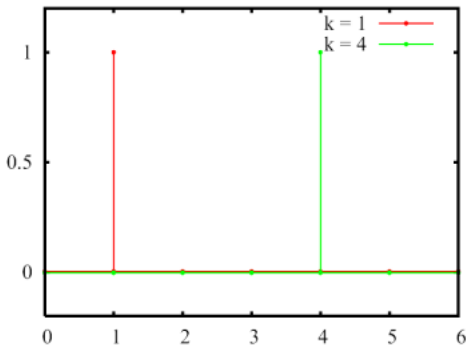


Figure 3.18: *Deterministic* distributions for two different mean values

A random variable with Erlang distribution of order r can be obtained as a sum of r exponentially distributed random variables with mean $1/r\alpha$ (see Figure 3.19). When the values of α and r are changed, the system will automatically recompute the mean = r/α and the variance $Var = r/\alpha^2$. A family of probability density functions that illustrate the impact of various (α, r) pairs is shown in Figure 3.20.

***Exponential* (exp(λ)) distribution**
This is a continuous probability distribution where $\lambda > 0$ is the distribution parameter, often called the *rate* parameter; the distribution is defined in the interval [0,∞). The probability density function is:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The exponential distribution is used to model Poisson processes. The time interval between two consecutive events generated by a Poisson process can be described by an exponential random variable with parameter λ. The parameter λ is a real number, the mean value is given by $1/\lambda$, and the variance is given by $1/\lambda^2$. A family of exponential density functions with different values of λ is shown in Figure 3.21.

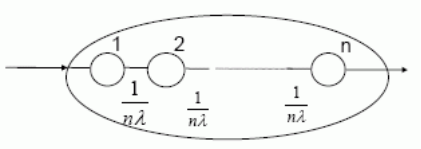


Figure 3.19: Simulation of an Erlang distribution with a sequence of exponential stages

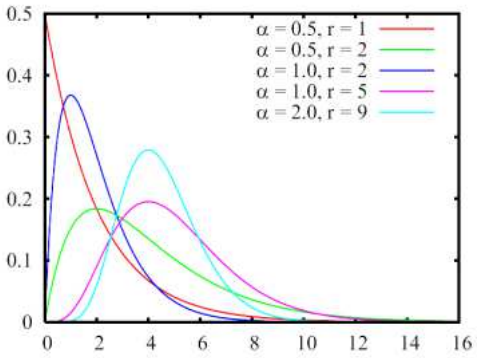


Figure 3.20: A family of Erlang distributions with different values of the parameters (α, r)

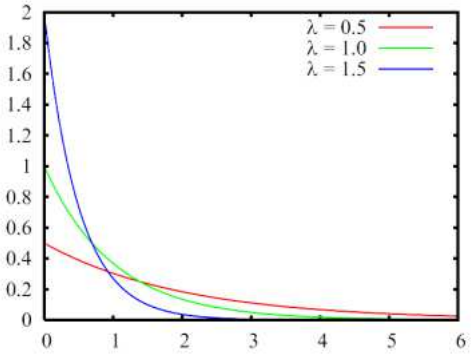


Figure 3.21: Exponential density functions with different mean ($1/\lambda$) values)

Gamma **(gam(α, λ)) distribution**
The *Gamma* distribution is a continuous probability distribution with two parameters: the shape α and the scale λ, both real numbers. The probability density function is:

$$f(x) = \frac{x^{\alpha-1}}{\Gamma(\alpha)\lambda^{\alpha}} \; e^{-x/\lambda}$$

where $\Gamma(t)$ is the Eulero function. When the Gamma distribution is selected, the user can either provide α and λ or the distribution mean equal to $\alpha\lambda$ and the variance $\alpha\lambda^2$. A family of probability density functions is shown in Figure 3.22, for a set of α and λ values.

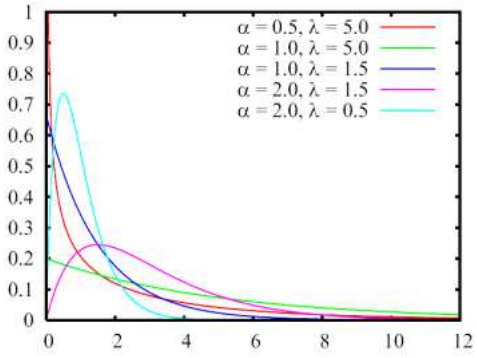


Figure 3.22: *Gamma* density functions for different values of α and λ

Hyperexponential **(hyp(p, λ_1, λ_2)) distribution**
The hyperexponential distribution describes a random variable characterized by a variability higher with respect to an exponential one with the same mean. It is the result of a weighted sum of two independent exponentially distributed random variables, with parameters λ_1 and λ_2 respectively. The weight p is the probability that the random variable behaves like the exponential variable with parameter λ_1 and *1-p* that it behaves like the

exponential variable with parameter λ_2. The probability density function is:

$$f(x) = p\ \lambda_1 e^{-\lambda_1 x}\ +\ (1-p)\ \lambda_2 e^{-\lambda_2 x}$$

When this distribution is used to model customer interarrival time or a station service time, with probability

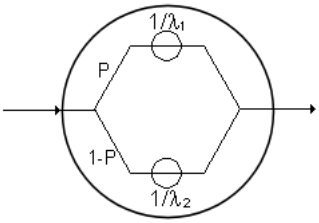


Figure 3.23: Generation of the values of an hyperexponential function through two exponential stages in parallel

p the next interval before a new arrival (or the next service time) is distributed like the upper server in Figure 3.23 the figure above while with probability *1-p* it will be distributed like the lower server. A family of hyperexponential density functions is shown in Figure 3.24. Note that when $\lambda_1 = \lambda_2$, the hyperexponential reduces to a simple exponential.

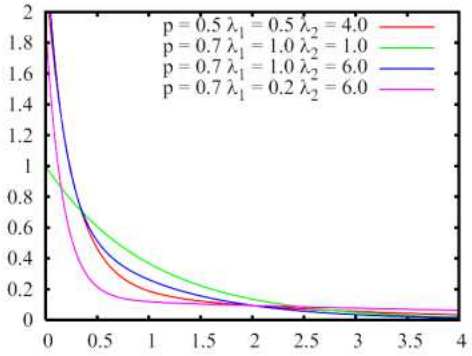


Figure 3.24: Family of hyperexponential density functions

***Lognormal* (lognorm(μ, σ)) distribution**

The lognormal distribution describes a random variable that may be seen as the result of exponentiation of a normal random variable with mean μ and standard deviation σ. A product of independent positive random variables also follows a lognormal law, a consequence of the central limit theorem in the log domain.

The probability density function of lognormal random variables is:

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right)$$

***Normal* (norm(μ, σ)) distribution**

The *Normal* distribution is also called Gaussian since its probability density function is the Gaussian function. It is well known for its bell-shaped density function.

The two parameters are μ = location (real number, it is the *mean* of the distribution), and σ = scale (real number, it is the *standard deviation* of the distribution, i.e., the square root of the variance). The density is symmetrical around the mean and the variance. The probability density function is:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}\ e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The *standard normal distribution* is the normal distribution with $\mu = 0$ (mean equal 0) and $\sigma = 1$ (variance and standard deviation equal 1). A family of normal density functions is shown in Figure 3.25. Note that the sequence generated is *not* perfectly normal since negative values are *discarded*.

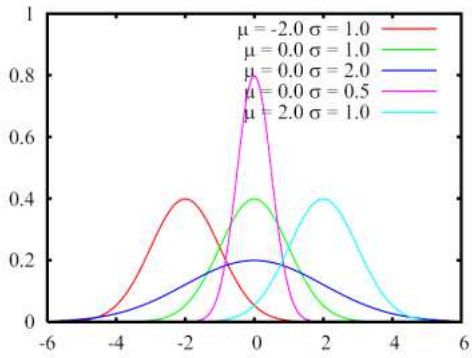


Figure 3.25: Family of normal density functions. The standard normal density is the green colored.

***Pareto* (par(α, k)) distribution**

This distribution is usually utilized to describe the behavior of social and economical phenomena (e.g., the distribution of wealth, where a small portion of the people owns the larger part of the wealth). It is characterized by the parameters $k > 0$, *location* (real), and $\alpha > 0$, *shape* (real). The probability density function is:

$$f(x) = \alpha k^{\alpha} x^{-(\alpha+1)}$$

If k and α are provided as input parameters, the simulator compute the mean $k\alpha/(\alpha - 1)$ for $k > 1$ and the variance

$$\frac{\alpha^2 x}{(x-1)^2(x-2)}$$

A family of Pareto density functions is shown in Figure 3.26.

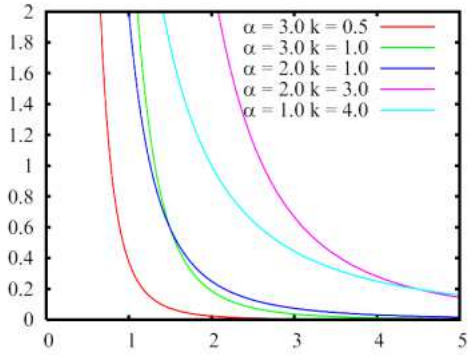


Figure 3.26: Family of Pareto density functions.

***Phase-Type* (ph($\boldsymbol{\alpha}, \boldsymbol{T}$)) distribution**

The *Phase-Type* (PH) distribution [Ste09] results from a sequential connection of Poisson processes, referred to as *phases*. Transition times between the phases are exponentially distributed, and only one phase is allowed to operate at a given instant of time. As a result, the phases can be mapped onto the transient states of a continuous-time Markov chain (CTMC) with a single absorbing state. The distribution itself will then correspond to a random variable representing the time that the CTMC spends travelling from an initial state to the absorbing one.

A PH(n) is composed of a total of n phases, and can be specified by a $1 \times n$ vector $\boldsymbol{\alpha}$ and an $n \times n$ matrix $\boldsymbol{T}$, that describe the initial probability of each phase and the transition rates between the n phases, respectively. The k-th moment of this distribution is obtained as

$$E[X^k] = k!\boldsymbol{\alpha}(-\boldsymbol{T})^{-k}\boldsymbol{e}$$

where $\boldsymbol{e}$ is an $n \times 1$ vector of all ones. Its squared coefficient of variation is given by

$$c^2 = 2E[X]^{-2}\boldsymbol{\alpha}(-\boldsymbol{T})^{-2}\boldsymbol{e} - 1$$

With a proper choice of parameters, the PH distribution can be used to generate a *wide range* of others, from the simplest exponential distribution, to more complex ones such as Coxian, Erlang, hypo-exponential and

hyper-exponential. For example, the 2-phase Coxian distribution shown in Figure 3.17 is exactly equivalent to a PH distribution with $\boldsymbol{\alpha} = [1, 0]$ and $\boldsymbol{T} = [-1, 0.125; 0, -0.125]$. Furthermore, the set of PH distributions is known to be dense in the field of all positive-valued distributions. That is, any positive-valued distribution can be approximated in arbitrary accuracy by the PH distribution.

***Replayer* (replayer("filename")) distribution**
When *Replayer* is chosen, a trace of data provided by the user can be reused as interarrival times or service times. The file format should be text/binary with values separated by CR (`Carriage Return`) or SPACE. The format of the values is DIGITS [POINT DIGITS], malformed elements will be ignored. In order to use this user-supplied file of data, the user should provide in the input window the absolute path name of the file. If the file does not exist or does not contain any valid value, JSIMgraph will use a default file (a file with all values set to zero).
Warning: be careful when using models generated in other computers that the path must be recognized also from the actual one.

***Uniform* (U(min, max)) distribution**
The *Uniform* distribution, also referred to as Rectangular distribution due the shape of its density function, describes a random variable that may assume all the values in the range (min, max) with the *constant* probability $1/(max - min)$. The probability is 0 outside the considered range.
The user can either provide the pair (min, max) or the *mean* $m = (max + min)/2$ and *variance* $c = (max - min)^2/12$. Two uniform density functions are plotted in Figure 3.27.

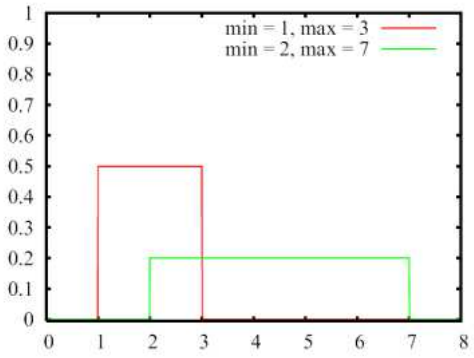


Figure 3.27: *Uniform* density functions with different range of values.

***Weibull* (weibull(λ, k)) distribution**
The Weibull distribution describes a non-negative random variable with probability density function

$$f(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} \qquad x \geq 0$$

This distribution is often used to model the time to failure of a system. A value of $k < 1$ (resp. $k > 1$) indicates a system where the failure rate decreases (resp. increases) over time. If $k = 1$, the failure rate remains constant over time.

The fitting of the Weibull distribution from moments is available only in approximate form. JMT implements an approximate method proposed by Justus at al. in 1977 as reported in [TGBB17, §2.1.3].

3.7 Performance indices

The following list includes the performance indices that can be obtained from a simulation run:

- **Arrival Rate:** the rate at which customers arrive to a station, i.e., the mean number of incoming requests in a time unit.
- **Balking Rate:** the rate at which jobs leave a queue (or the whole network) due to balking, i.e., a decision to leave prior to joining the queue in question.
- **Drop Rate:** the rate at which the customers are dropped from a station or a region for the occurrence of a constraint (e.g., maximum capacity of a queue, maximum number of customers in a region). At the system level it refers to the rate at which the customers are dropped from the system.

- **Earliness:** For a job with deadline d that completes at time t since entering the network, the earliness is $\max(0, d - t)$.
- **Effective Utilization:** The utilization of jobs (or a class thereof) at a station ignoring setup costs.
- **FCR Total Memory:** It describes the total memory of the customers in the Finite Capacity Region. Each job has a memory attribute and this is a secondary constraint in addition to the capacity constraint.
- **FCR Total Capacity:** It describes the total capacity of the customers executed in the Finite Capacity Region
- **Firing Throughput:** rate at which a *transition station* fires, i.e., the number of firings occurring in a time unit.
- **Fork Join Number of Customers:** mean number of customers (job) in a Fork/Join region, either waiting in the entry buffer *or* staying inside region.
- **Fork Join Response Time:** average time spent by a customer (job) in a Fork/Join region. The synchronization time of the tasks at the Join is included.
- **Lateness:** For a job with deadline d that completes at time t since entering the network, the lateness is $t - d$.
- **Number of Customers:** It can be defined at the station or system level. At the station level, it refers to both customers waiting in the queue *and* those receiving service. At the system level, this metric shows the global number of customers in the system. These values are described per each class of customers.
- **Queue Time:** average time spent by the customers waiting in a station queue. It does not include the Service Time.
- **Reneging Rate:** rate at which jobs leave a queue (or the whole network) due to impatience upon waiting in the buffer(s).
- **Retrial Orbit Size:** number of jobs retrying to join a station
- **Retrial Rate:** rate at which jobs retry to join a station
- **Retrial Residence Time:** mean time that jobs spend in a retrial orbit before being able to re-join the station
- **Residence Time:** total time spent at a station by a customer, both queueing and receiving service, considering *all* the visits at the station performed during its complete execution.
- **Response Time:** It can be defined at the station or system level. At the station level it refers to the average time spent at that station by a customer for a *single* visit (sum of Queue time and Service time). At the system level, it refers to the average time a customer spends in the system in order to receive service from the various stations it visits. It corresponds to the intuitive notion of response time, as the interval between the submission of a request and the reception of the response. It may be obtained summing the *Residence Times* of all the resources. These values are described per each class of customers.
- **Response Time per Sink:** average time spent in system by a customer before dropping at the selected sink.
- **Power:** The optimal operational point of a system is the point corresponding to the maximum System Throughput X with the minimum System Response time R, i.e., the value of the ratio X/R is maximized in this point. This ratio is known as `System Power`. Typically is computed at the level of the entire system. The `System Power` may be computed also for each class of customers considering the throughput and the response time of the class (in this case is referred to as `Per-class System Power`.
- **Tardiness:** For a job with deadline d that completes at time t since entering the network, the tardiness is $\max(0, t - d)$.
- **Throughput:** At the station level it refers to the rate at which customers departs from a station, i.e., the mean number of requests completed in a time unit. At the system level it refers to the rate at which customers departs from the system. These values are described per each class of customers.
- **Throughput per Sink:** rate at which customers departs from the system with respect to the selected sink, i.e., the number of requests completed in a time unit that reach a given sink. These values are described per each class of customers.
- **Utilization:** percentage of time a station is used (i.e., *busy*) evaluated over all the simulation run. It ranges from 0 (0%), when the station is always *idle*, to a maximum of 1 (100%), when the station is constantly *busy* servicing customers for the entire simulation run. *Queueing stations* may have more than one server, their number is a parameter to be specified (*default* is 1). It is important to point out that the *utilization* U of a queueing station with a *single server* is given by $U = \lambda S$, and at a station with

mservers the utilization of *any* individual server is given by $U = \lambda S/m$. In *delay stations*, for consistency with Little's law, the *utilization* is computed as the *average number of customers* in the station, and thus it may be *greater than 1*.

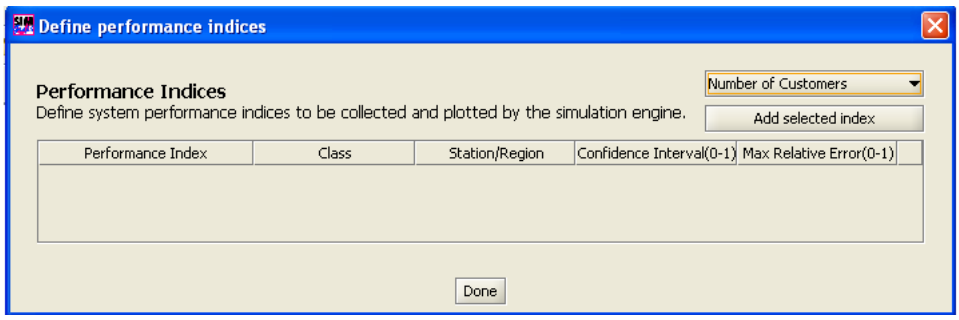


Figure 3.28: The window for the selection of the `performance indices`.

Any subset of indices from the above list can be plotted as model output. Each index is associated with a class and a station and will be computed within the given Confidence Interval and Maximum Relative Error, both defined on the (0-1) range, by performing the following steps:
1. Select the index you want to add to the model from this menu:

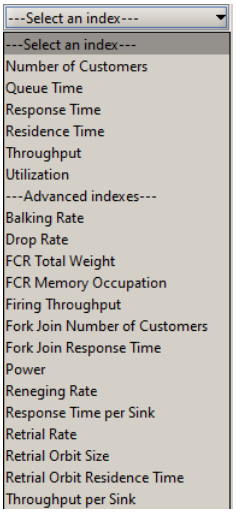


Figure 3.29: The drop-down list for the selection of the *performance indices*.

2. Select `Add Selected Index` and the index will be added to the panel. Then the index must be set.
3. Select from the `Class` menu a `Single class`, or `All Classes`, for which the index must be computed.

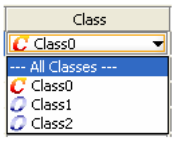


Figure 3.30: The drop-down list for the selection of the `Class`.

4. Select the *Station* for which the index must be computed from `Station` menu. In case of system wide indices, namely, System Throughput and System Response Time, this option is not available.

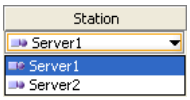


Figure 3.31: The drop-down list for the selection of the `Station`.

5. Double click to modify the default values for the *Confidence Interval* size of the solution and for the *Max Relative Error* of the greatest sample error, if you want more/less accurate results.

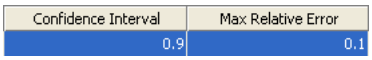


Figure 3.32: Definition of the `Confidence Interval` and `Max Relative Error`.

6. Repeat these steps for all the indices you want to include in the model output.

NOTE: Users must be informed that the value of *System Power* computed by JSIM*graph* and JSIM*wiz* is approximate and not the exact X/R where System Throughput X and System Response Time R are obtained from the simulator. This is mainly due to the implementation limitations and the reduction of the computational complexity.
NOTE: Errors in parameter settings will be detected only when the simulation is started, raising a warning or an error message.

3.7.1 Confidence Intervals

The confidence interval shown at the end of a simulation run contains the true value of the estimated index with the selected probability (1-α) or, equivalently, if an experiment is repeated many times, in (1-α)*100% of cases. Various difficulties in meeting theoretical assumptions can cause that the real percentage of the confidence intervals containing the true parameter differs significantly from (1-α). The robustness of the above methods of data collection and analysis is usually measured by the coverage of confidence intervals, defined as the frequency with which the intervals (X(n)-Δx, X(n)+Δx) contain the true parameter value μ_x, at a given confidence level (1-α), 0< α <1. The coverage analysis can be applied only to systems with a theoretical well-known behavior,

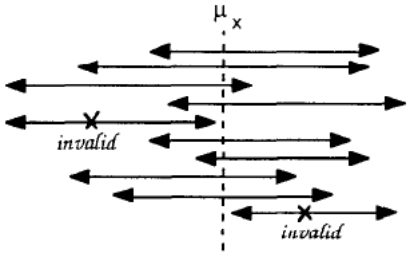


Figure 3.33: *Confidence intervals* for the correct parameter value μ_x evaluated in 10 runs. The coverage is 80% since two of them do not include the correct value.

since the value of μ_x has to be known. Any analyzed method must be applied in a statistically significant number of repeated simulation experiments, usually 200 or more replications, to determine the fraction of experiments producing the final confidence intervals covering the true mean value of the estimated parameter.

3.7.2 Max Relative Error

Let x be the true value of a quantity and x_i be the measured or inferred value. The relative error is defined by:

$$\frac{|\overline{X}(n) - \mu_x|}{|\mu_x|} \quad where \quad \overline{X}(n) = \sum_{i=1}^{n} \frac{x_i}{n}$$

where μ_x is the mean value. The relative error is the sum of the deviations between the sample values and the mean value, divided for the mean value.

3.7.3 Advanced Metric Configuration

As show in Figure 3.34, for certain advanced simulation features it is possible to further configure the performance indices. Upon clicking on the *Edit* button on the right of each metric, a dialog window allows restricting the computation of that metric to specific server types, in situations where a queueing station has been configured to allow service compatibilities. We point to subsection 3.11.3 and Figure 3.65 for additional details on the latter feature.

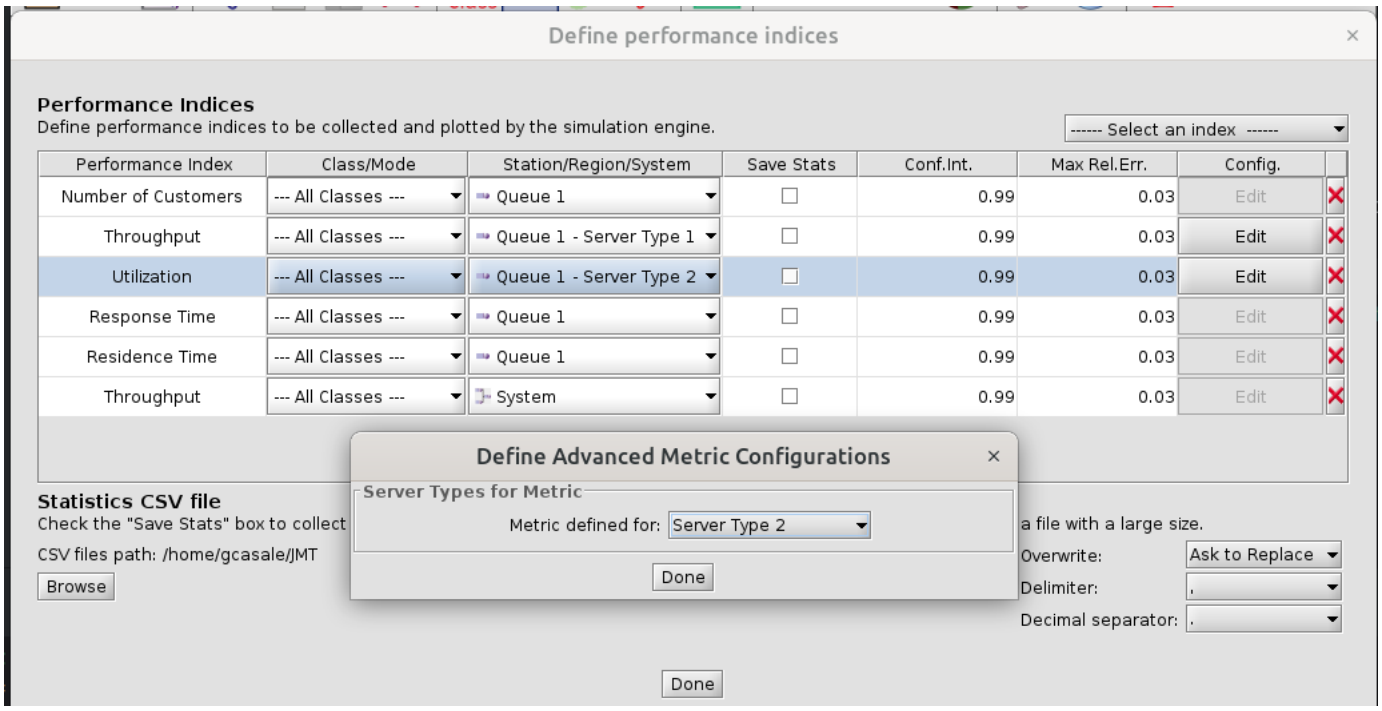


Figure 3.34: Window for `Advanced Metric Configuration`

3.8 Simulation Parameters

The `Define Simulation Parameters` window (see Figure 3.35) can be reached either by selecting `Simulation Parameters` from the `Define` menu or by clicking the `Define Simulation Parameters` icon from the toolbar. The *default* values of Simulation parameters are shown in section 3.12.3 and section 4.4. The

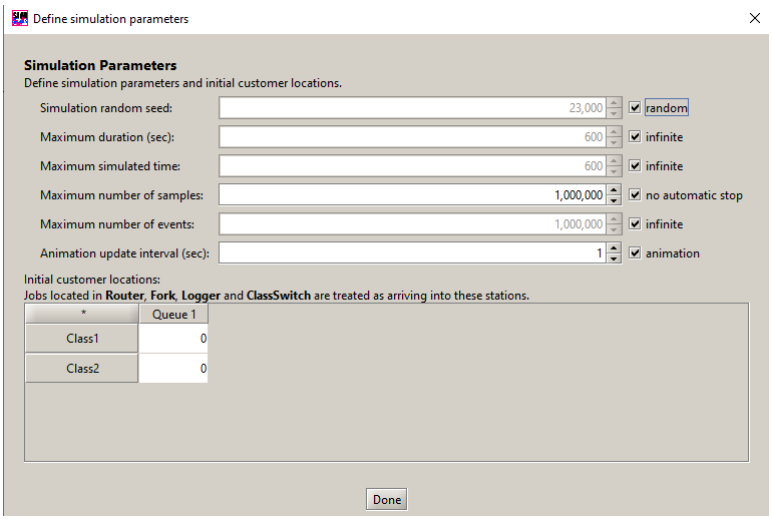


Figure 3.35: Window for the definition of the `Simulation Parameters`

parameters that can be used to control the simulation are:

- **Simulation Seed:** is the number used by the simulation engine to generate pseudo-random numbers. If you change the seed, you will obtain a different sequence of pseudo-random numbers. You may select `random`, which indicates that the simulator will use a seed randomly generated, or you may type the number that you want to be used as seed. When a simulation is repeated using the *same seed*, the same sequence of pseudo-random numbers will be generated, thus leading to *identical results*. The default value is custom simulation the one which has been set in the `Simulation parameters` section of the `Editing Default Values`. In the figure it is required the generation of a random seed.
- **Maximum duration (wall time, sec):** It represents the maximum amount of time, in seconds, that

the simulation will run, i.e., the *wall time* value. If the simulation ends before the maximum duration time (due to the automatic stop controls), this parameter is ignored. The *default* value is `infinite`, deselect it and specify the maximum time value if you do not want the simulation to run for a possibly very long time. In this case, the simulation stops when the time limit is reached, although a reliable solution may not be available yet.

- **Maximum simulated time:** It is the maximum simulated time allowed for the current simulation run. The simulation will stop when this time is reached (if other stopping criteria eventually activated are not reached). In the example of Figure 3.35 the simulation time is set to 200,000 time units.
- **Maximum number of samples:** It is the maximum number of samples (data) for each index that JSIM*graph* collects before ending the simulation. The default value for the `maximum number of samples` is 1,000,000; you may increase it (for a more accurate simulation) or decrease it (for a faster simulation).
- **Animation Update Interval (sec):** This is the granularity at which results are plotted on the screen, i.e., the time interval (we will refer to the *wall time* and not to the simulated time) between the plot of two consecutive values of the graphs, as the simulation proceeds. An `animation` checkbox is used to enable or disable plot animation during the simulation run.

A simulation run is *stopped* in case of:
- *Success*, if the values of the considered index have reached the required `Confidence Interval` with an error less than the `Max Relative Error`
- *Failure*, if the simulation has analyzed the `Maximum number of samples` but has not reached the required `Confidence Interval` or the error is greater than the `Max Relative Error`
- *Failure*, if the `Maximum duration time` (*wall time*) has been reached before the *Success* criterion
- *Failure*, if the `Maximum simulated time` has been reached before the *Success* criterion.

Initial state of a simulation

The *initial state* is the model state at time 0, typically described by the number of customers present in each station at the beginning of the simulation. Without a priori knowledge, these customers are initialized as if they enter the model simultaneously at that time. In the case of a *multiclass* workload, different classes of customers are put into the stations according to the natural order (starting from the *first* class). The *priority* assigned to each class and the *queueing policy* implemented by each station are taken into account during the procedure.
For *closed* classes, all customers are allocated by *default* to their *reference stations*. This arrangement can be modified as long as the total number of customers remains the one defined in the `Classes` tab. For *open* classes, it is possible to preload each station with any desired number of customers. In Figure 3.35, the initial state of a model with four stations and two closed classes (*Class1* with 3 customers in the *CPU* station and *Class2* with 10 customers in the *Users* station) is shown.
Note that the selection of the initial state is important since, as a factor of the model complexity, it may influence the transient period and the convergence time of some indexes.

3.9 What-If Analysis

A What-If Analysis consists of a series of simulations in which one or more *control* parameters are varied over a specified range. This allows the observation of system behavior under a spectrum of conditions, unlike the single simulation run where the system is observed under a fixed set of parameters. By default the What-If Analysis is not enabled. It must be activated explicitly and its parameters should be defined.
Activating the What-If Analysis
After completing the definition of the model and selecting the `What-If Analysis` tab, check the `Enable what-if analysis` checkbox to activate it.
Selecting the What-If Analysis
After enabling the What-If Analysis, it is possible to select the parameter to control the sequence of simulation runs using the menu of Figure 3.37. When you select a What-If Analysis parameter, the bottom section of the window of Figure 3.36 will change depending upon the parameter. In the right portion a `Description` is provided of the selected parameter and of the impact of its variation. In the left portion the details of the parameter range (`From` and `To` fields) and the number of executions (`Steps`) on the range are provided. The set of modifiable parameters depends upon the number and type of classes in the model: in a *single class* model, you simply select the parameters you want to change during simulation while in a *multiclass* model, you must select the parameter and specify whether you want to apply the variation to all the classes or just to one specific class.

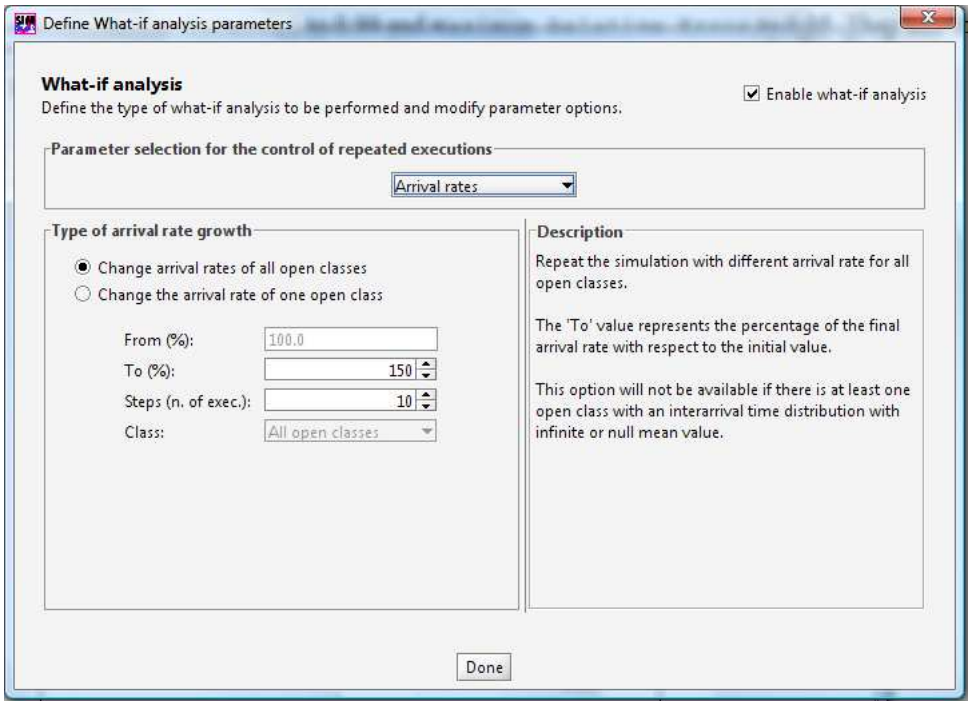


Figure 3.36: The window for `What-If Analysis` definition.

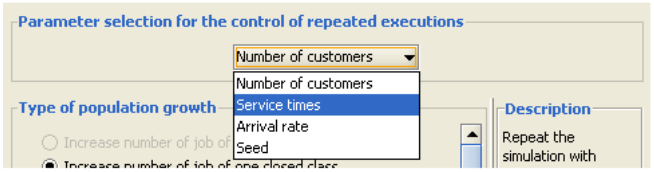


Figure 3.37: Selection of the parameter for the control of a What-If Analysis.

The parameters that may be used to *control the sequence of executions* in a What-If Analysis are:

- **Number of Customers:** (only for models with **closed** classes)
  JSIM repeats the simulation changing the number of customers in each run, starting from the number inserted in the `From N` field, to the value inserted in the `To N` field. The simulation is repeated `Steps (no. of exec.)` number of times. In Figure 3.38 a What-If Analysis is planned on a single closed class model. The option `Increase number of jobs of one closed class` has been selected. The initial value of the number of customers is not modifiable from this window, as it is part of the `Classes` tab. The final value should be specified in the field `To N`. In this case, with a final value of 20 and 6 Steps, simulations are run for 10, 12, 14, 16, 18 and 20 customers, respectively. Only the customer number in the class selected in the `Class` (*ClosedClass*, in the picture) will be changed. The remaining classes (if they are present) will keep their initial number of customers. The simulator control that the sum of the *percentages of jobs* of the various classes (represented by the components β_i of vector β) evaluated over the global population N of the model add up to 1, as shown in the `Population mix` table on the bottom of the window of Figure 3.38 in the simple case of a single class model.
  If the `Increase number of jobs of all closed classes` option is selected, the overall population is increased keeping constant the relative proportion of jobs in the various classes, i.e., the values β_i (also referred to as *population mix*). Because in the JSIM the number of customers in each class can only be integer numbers, only the population vectors with all integer components can be considered. Therefore, the actual number of executions may be smaller than the one specified.
  The population mix describes the way the global population is subdivided between the classes, i.e., the percentage of customers in each class (β_i values) over the total population. The population mix can be modified extensively selecting the way we want to change it with the increasing of the global population of customers: all classes increase proportionally or only one class increases keeping constant the jobs of the other classes (select the option `Increase number of jobs of one closed class` in this case). Mixed models can be analyzed too. Remember that only models with closed classes (at least one) will be considered in this What-If Analysis.

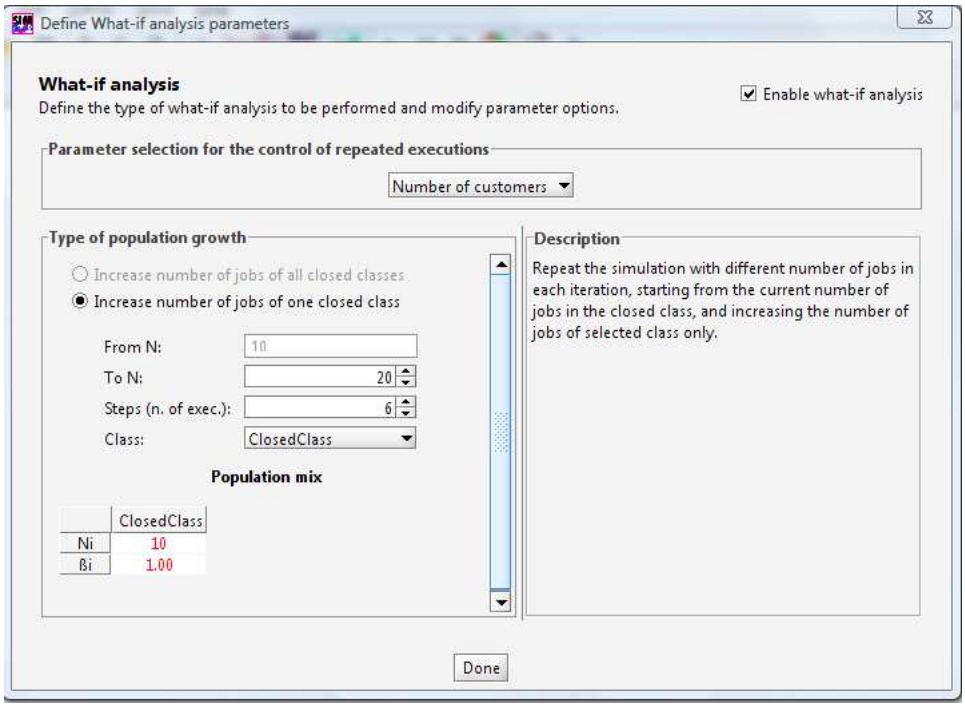


Figure 3.38: Selection of the `Type of population growth` in a What-If Analysis.

- **Arrival Rate:** (only if there are **open** classes)
  Arrival rate is the frequency at which jobs arrive at a station during a period of time. Similarly to the number of customers, the arrival rate can be changed for one specific open class or for all the open classes in the model.
  If the option `Change the arrival rate of one open class` is selected, the final arrival rate and the number of steps must be specified.
  The initial arrival rate is specified in the `Arrival Rate` section of the `Classes` tab and it is not modifiable here.
  If the option `Change arrival rates for all open classes` is selected, the final value is expressed as a percentage of increase with respect to the actual value that is applied to the arrival rates of all the classes. Figure 3.36 shows the settings for a What-If Analysis of the arrival rate of all the open classes. The `To` field is set to 150%, which means that for each class the final arrival rate will be 1.5 times greater than the initial value. 10 runs will be executed with equally spaced (in percentage) intermediate values of arrival rates.
- **Total capacity:** Allows to repeat the simulation altering the total capacity of a given station. This is the total number of jobs in the station including the one(s) on service and corresponds to the Capacity panel of a queue parameters panel (Queue Section). If the queue at start is configured with infinite capacity, note that at the end of the What-If simulation it will be changed into the main model to have a finite capacity.
- **Service time:** (for **all types** of classes) Service Time is the time required by a customer at each visit of a station. In this case, besides the final value and the number of runs to be executed, the station and the customer's class (or all the classes) whose service time will be varied must be specified. Figure 3.39 shows

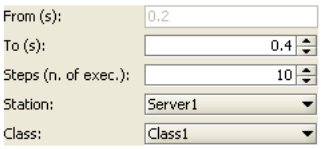


Figure 3.39: Selection of the final value of `service time` in a What-If Analysis.

the settings for a What-If Analysis of the service time of class *Class1* at station *Server1*. The service time distribution will not change. Only its average will be modified to span the range defined by the initial value (specified in the `Station Parameters` tab) and the final value specified here in the `To` field.
If the `Change service time of all classes` option is selected, the range of service time to explore is

expressed in percentage, starting from the initial value specified at model definition (which is considered as 100%). The station whose service time should be modified must be specified.

- **Routing probability:** One of the most advanced What-if analysis is the variation of the routing probabilities out of a station. The analysis option is enabled only if one or more stations in the model use the *Probabilities* routing policy.
  The associated What-if panel is shown in Figure 3.40 In the panel, the Source Station, Destination

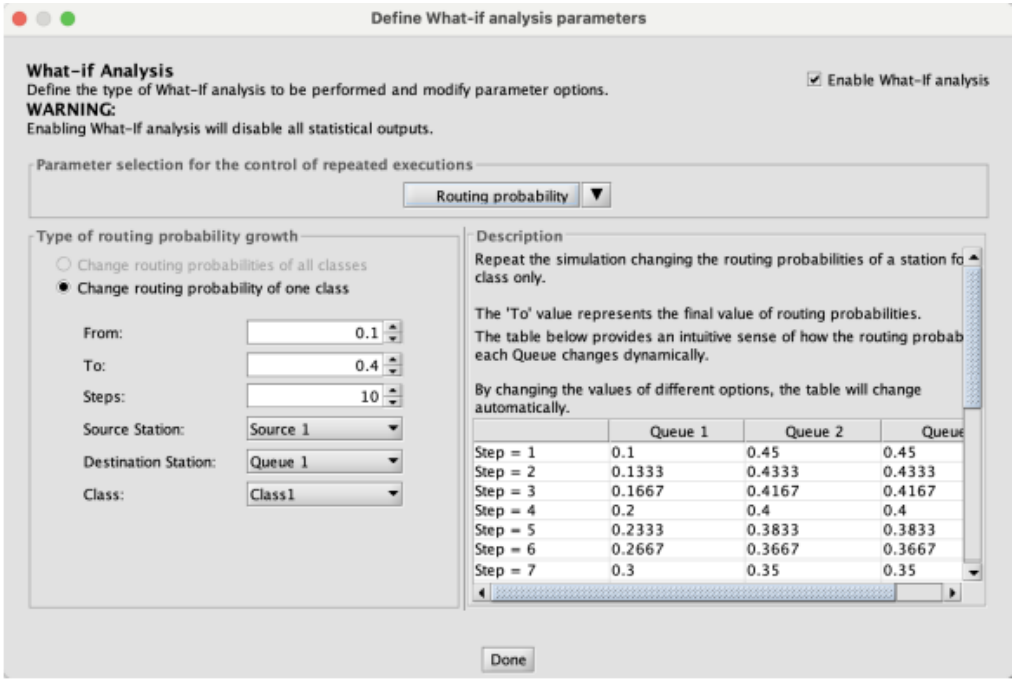


Figure 3.40: What-if analysis on routing probabilities.

Station, and the Class fields identify a specific routing probability that needs to be verified in the given range (From/To/Step). In general, the same station may have other probabilities for jobs of the same classes. These are rescaled during the What-if to maintain the same relative ratio at all steps. For example, assume that a class has three routing probabilities out of Station 1 called p_1, p_2 and p_3.
Assume that $p_1 = 0.4$, $p_2 = 0.3$ and $p_3 = 0.3$. Then, if we vary p_1, the value of p_2 and p_3 will be simultaneously varied so that $p_2 = p_3$ at every step. Similarly, if $p_1 = 0.1$, $p_2 = 0.6$ and $p_3 = 0.3$, then as we vary p_1 JMT will always ensure that the new values of the other probabilities satisfy $p_2 = 2p_3$. A list of the routing values obtained in this fashion is given in the preview box on the right.

- **Seed:** (for **all types** of classes)
  Unlike the previous cases, where the analysis is performed for a range of values of one of the model parameters in order to investigate the system behavior under a variety of conditions, a What-If Analysis on the seed aims at evaluating the sensitivity of the simulation engine to numerical conditions, in particular to the numerical value used by the pseudo-random number algorithm to generate the sequence of numbers, i.e., the seed. The simulation engine uses a pseudo-random number generator to generate the sequence



Figure 3.41: Selection of the number of repeated executions with different `seeds` in a What-if analysis.

of values used in each execution. By changing the seed, a different sequence of values is produced, thus leading to different numerical results. The first seed is the one defined in the `Simulation Parameters` tab. The number of executions is specified in the panel of Figure 3.41 and the simulation engine generates as many pseudo-random seeds to be used (one for each execution).

3.10 Finite Capacity Region (FCR)

A `Finite Capacity Region` (FCR) is a region of the model in which the number of customers that can be admitted at any given time is controlled by two parameters, called *capacity* and *memory*, which have similar properties.

Capacity is a weighted sum of the number of customers in the FCR, where the weight of a job is given by its *job capacity* value. Each job admitted in the FCR may additionally carry a *job memory* requirement and therefore will be subject to a second constraint in terms of total available memory. The two constraints are therefore conceptually similar, but there exist models in which both are applied simultaneously with different upper limits. Moreover, they can be recorded in separate performance indices.

It is possible to define constraints on capacity and memory at *two levels*:

- `Global (the entire workload)`: an upper bound for the number of customers (or a limit to their memory occupancy) that are in the region, regardless of the classes they belongs to.
- `Specific (per class)`: an upper bound for the number of jobs (or a limit to their memory occupancy) for a specific customer class in the region.

If the FCR region is full of jobs (or a certain class of jobs), to avoid violating the constraints, the following jobs will be blocked (*queued* or *dropped* before entering the region, according to the policy selected, see Fig.3.44). During the simulation the most restrictive constraint (Global or Specific) is applied. For example, consider a multiclass model with two classes (*Class1* and *Class2*) with a Global constraint of 100 customers, a Specific constraint of 30 customers for *Class1* and no Specific constraint for *Class2*. If during the simulation the FCR contains 30 customers belonging to *Class1* and 60 customers belonging to *Class2*, a new incoming job of *Class1* will be rejected (as it's Specific constraint will be violated) while a new incoming job of *Class2* will be accepted (as it will not exceed the Specific constraint). In case in which 20 jobs of *Class1* and 80 jobs of *Class2* are already inside the FCR, a new incoming job will always be rejected, despite of its class, as it violate the *Global* constraint.
The `utilization` of an FCR is set to 0 since, as a function of the stations allocated inside the region, it may have no meaning.

How to define an FCR:
1. Select the stations to include in the FCR with the mouse: left-click with the mouse on each station while holding down the shift key. The selected stations are framed with green dashed lines, as shown in Figure 3.42.
2. Select the icon  and the Finite Capacity Region will appear with a blue background, see Figure 3.43.

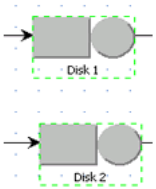


Figure 3.42: Selection of the stations to be included in an FCR.

3. Double click on `FCRegion` to set the properties. The properties panel will appear, Figure 3.44.

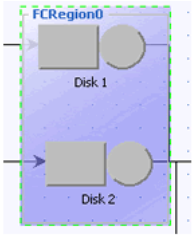


Figure 3.43: The area with the blue background is the FCR.

Global Properties
`Region Name`: the name of the region created
`Region Max Capacity`: max number of customers that can be in the region. Enable *Infinite* if you do not want a bound
`Region Max Memory`: it is possible to define the *maximum* memory size required by the jobs that are in the region. Enable *Infinite* if you not want a bound
`Region Groups`: when this feature is *Enabled* the classes of the jobs can be grouped according to the objectives of the study.

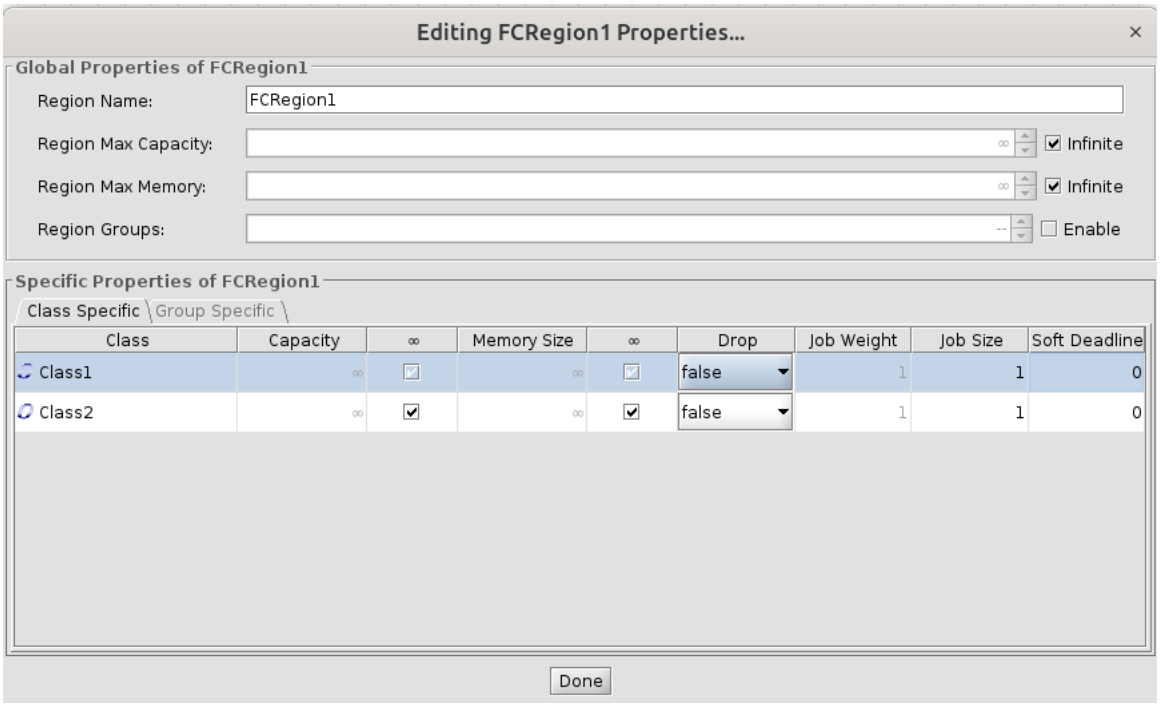


Figure 3.44: Window for the definition of the FCRegion properties.

Class Specific Properties
In the bottom part of the panel, the *class specific properties* can be defined:
`Max Capacity`: maximum number of customers of that class into the FCR (*Infinite* if no bound is set)
`Max Memory`: maximum memory occupied by the jobs of that class that are in the region (*Infinite* if no bound is set)
`Drop`: when the FCR has exhausted capacity or memory, and `Drop` is `false`, the incoming jobs are not dropped but are queued outside the region, until the FCR can accept them. If `Drop` is `true`, the jobs are dropped from the system
`Job Weight`: weight assigned to each job of that class towards computing the total capacity.
`Job Size`: weight assigned to each job of that class towards computing the total memory.
`Soft deadline`: relative deadline for the response time from entering the FCR to leaving it. If missed, the job continues executing.

3.11 Type of Stations

To define a new model select `New` from `File` menu or click the icon  and draw the network topology.

Selecting the stations
From the second toolbar (Figure 3.45), select a station and click on the model panel to insert it (drag and drop). The available stations are:



Figure 3.45: The toolbar for the selection of the stations to be inserted in the model.

 Insert a new Source
 Insert a new Sink

Insert a new Router
Insert a new Delay
Insert a new Queue
Insert a new Fork
Insert a new Join
Insert a new Logger
Insert a new ClassSwitch
Insert a new Semaphore
Insert a new Scaler
Insert a new Place
Insert a new Transition
Add selected stations to a new finite capacity region

To rotate selected stations press one of the or or buttons. To optimize the final layout of the network (only if implemented with the old interface) press the button.

Connecting two elements
To link two stations of the model:

1. Select the icon for the automatic connections or for the arc connections (Bezier curves)
2. Select and hold the mouse pressed from one station to the other station to which you want to connect to.

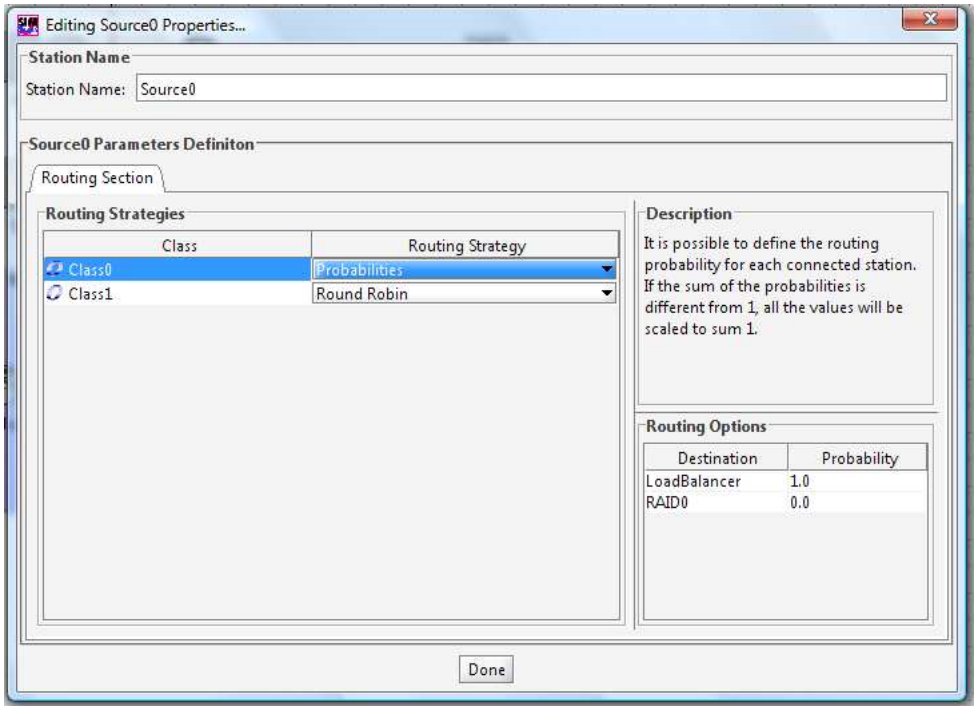


Figure 3.46: Window for `Editing` the properties of a `Source` station.

3.11.1 Source Station

Setting station properties
Open classes are characterized by an infinite stream of jobs that can enter the system. Source stations are used to generate customers in the model. The interarrival time of the customers of each class should be defined as a parameter of the class. The routing strategy defines the first station a newly created customer will visit. Only open class customers can be routed from source stations. Note that the `Load Dependent Routing` cannot be selected in a `Source` station since it has no meaning for this type of station. It is listed among the policies of the `Source` station only for convenience and completeness of the list of the routing policies.

Setting or changing the properties
Double click on the station icon to open the properties panel of Figure 3.46. There is only one section: `Routing Section`.

Routing Section
In the routing section, for each class, the generated customers are routed to the devices connected to the analyzed station according to various routing strategies. The following algorithms are available :

- **Random:** Customers are routed randomly to one of the stations connected in output to the considered station. The outgoing links are selected with the same probability. Figure 3.47 illustrates the routing strategy with 3 output links. For each link the probability to be selected is 1/3.

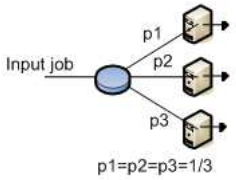


Figure 3.47: Routing with *Random* algorithm.

- **Round Robin:** Customers are cyclically routed to the outgoing links according to a circular routing. As shown in Figure 3.48, the first customer is sent to the top station, the second customer is sent to the central station, and the third customers is sent to the bottom station. The next customers would be sent to the top station again, and so on.

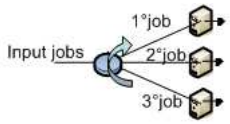


Figure 3.48: Routing with *Round Robin* algorithm.

- **Probabilities:** The routing probability for each outgoing link must be defined. The sum of all probabilities must be equal 1 (Figure 3.49 and Figure 3.50). If the values provided do not satisfy the constraint, JSIM automatically normalizes the values before the simulation starts. The probability for each output link may be set in the panel on the bottom right of the window shown in Figure 3.46.



Figure 3.49: Definition of the probability of the outgoing links.

- **Join the Shortest Queue (JSQ)**: (sometimes referred to as *Shortest Queue Length*) Each customer is routed to the station connected in output that has the *smallest* number of customers either in queue and in service at the time the customer leaves the routing station. Figure 3.51 shows a case where the number of customers (in queue and in service) at the devices are 3, 2, and 1 job, respectively, from top to bottom. The exiting customer will be routed to the bottom station, since its queue is the shortest (1 customer).
- **Shortest Response time:** Customers are sent to the station where the response time for the job's class is the smallest at the moment a customer leaves the routing station. Figure 3.52 shows that at the time of routing, the middle station has the smallest response time, R, so the customer will be sent to it.
- **Least Utilization:** The destination station is chosen as the one with the smallest utilization at the time routing is performed. In the example, shown in Figure 3.53 depicted in the picture, the top station is the least utilized, so it will receive the next customer to leave the blue station.
- **Fastest Service:** A customer is routed to the device with the smallest service time, S, for the job's class. In Figure 3.54 the exiting customer will be routed to the top station since its service time is the minimum among the three.

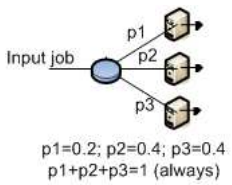


Figure 3.50: Routing according to the *Probability* algorithm.

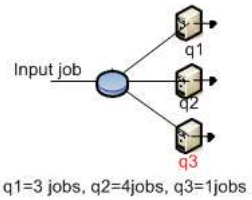


Figure 3.51: Routing according to the *Join the Shortest Queue* algorithm.

- **Load Dependent Routing (cannot be selected in a Source station):** The customers are sent to the destination station according to both the *probabilities* defined by users *and* the *load* of the source station (i.e., the station that we are considering). The routing decisions are made independently for each class. The *load dependent* routing probabilities must be described for *all the classes* by the users. Fig.3.55 shows the selection of the `Load Dependent Routing` policy for the *Class1* customers. The definition of the routing probabilities of the station `Queue 1` for *Class1* customers is shown in Fig.3.56. When the number N_{Q1} of *Class1* customers in station `Queue1` is $1 \leq N_{Q1} \leq 4$ the *Class1* customers are routed to station `Queue 2` with probability 0.1 and to station `Queue 3` with probability 0.9. When the number N_{Q1} of *Class1* customers in station `Queue1` is $5 \leq N_{Q1} \leq 9$ they are routed with the same probability 0.5 to `Queue2` and `Queue3`. When their number in station `Queue1` is $N_{Q1} \geq 10$ they are routed to station `Queue 2` with probability 0.8 and to station `Queue 3` with probability 0.2.
- **Power of k choices (with and without memory):** as opposed to picking the minimum queue length (JSQ) across all n connected stations, this policy queries the lengths of a random `k` $< n$ of them. The *shortest* amongst these `k` queues is then picked and jobs are routed to that station. If `Enabled Memory` is flagged, the queue lengths of the `k` queues are recorded in a local *memory* array. Jobs are routed to the station with the smallest queue length in the memory (number of jobs).
- **Class Switch:** upon probabilistic routing a job to a destination, this policy also permits to change its class.
- **Disabled:** This is simply a bookmark. If the user knows that jobs of a class cannot be routed to this station, this strategy can be used to indicate that routing is left unspecified.

3.11.2 Sink Station

Open class customers leave the system once they have received all the services they need. Sink stations are used to model customers leaving the system, as they enter the sink station but do not ever leave it. Sink stations have no parameters, only incoming connections from one or more stations, depending upon the model.

3.11.3 Queueing Station

The *Queueing Station* is one of the most important components in a queueing network model. A queueing station consists of two main components: a *queue* and a *server* (one or more) to execute the requests. If all

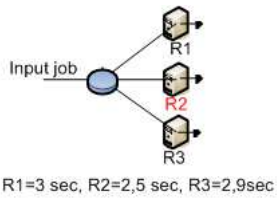


Figure 3.52: Routing according to the *Shortest Response Time* algorithm.

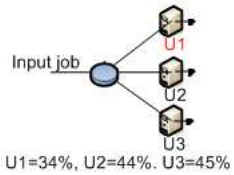


Figure 3.53: Routing according to the *Least Utilization* algorithm.

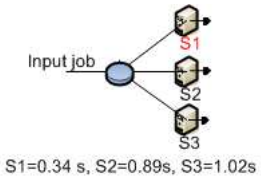


Figure 3.54: Routing according to the *Fastest Service* algorithm.

the servers of the station are busy when a new customer arrives at the station, the arriving customer join the queue and waits its turn to receive service from the first idle server. The queueing discipline determine which customer is served next when a server becomes free. The response time at queueing stations includes service time and queueing time. Queueing stations may have more than one server, their number is a parameter to be specified (*default* is 1). It is important to point out that the *utilization* U of a queueing station with a *single server* (i.e., the fraction of time in which the server is busy) is given by $U = \lambda S$, and at a station with $mservers$ the utilization of *any* individual server is given by $U = \lambda S/m$.

Set or change of the properties
Double click on the `Queueing Station` icon and the `Editing Server Properties` window will open (see Figure 3.58).

`Station name` is the name of the station that the user like to assign. There are three tabs: `Queue Section`, `Service Section` and `Routing Section`.

Queue Section
The Queue section allows the specification of the queue *capacity* (finite or infinite) and scheduling *policy*. Different classes of customers may have different scheduling policies associated with them.

- **Capacity:** a station can accept any customer number and let them wait in queue, in which case its capacity is considered infinite, or it can only accept a finite number of customers. In this case its capacity is finite, the max number of customers is to be specified in the field `length`.
- **Queue policy:** it is the algorithm used to decide which customer to serve next. A variety of factors can contribute to the order in which customers are served, such as arrival order, priorities associated with a class, the amount of service already provided to customers, etc. The scheduling disciplines are grouped into three categories: `Non-preemptive Scheduling`, `Non-preemptive Scheduling with Priority` and `Preemptive Scheduling`.
  As shown in Figure 3.58, the following `Non-preemptive Scheduling` disciplines (with or without `priority` selection) are supported:
  - **FCFS:** Customers are served with the same sequence in which they arrive at the station. If the model is exported to MVA, the following constraint is enforced in the exported model because the MVA analytical algorithm (see the BCMP theorem in [BCMP75]) requires in a multiple class environment the customers of all the classes must have the same average service time $S_{c,k}$ at a FCFS queue station. In order to take care of the differences among the service requirements of the different classes, the total number of visits to the station per each class $V_{c,k}$ is adjusted in order to obtain the correct values of service demands $D_{c,k}$. If a user is interested only in simulation (with JSIMgraph or JSIMwiz) no constraint is imposed on the parameters regarding service times, visits, and service demands.
  - **FCFS (Priority)**: Customers are ordered according to their arrival time but customers with higher priority jump ahead of customers with lower priority (a *small* priority number means low priority). Customers with the same priority are served FCFS.
  - **LCFS:** An arriving customer jumps ahead of the queue and will be served first. The LCFS discipline

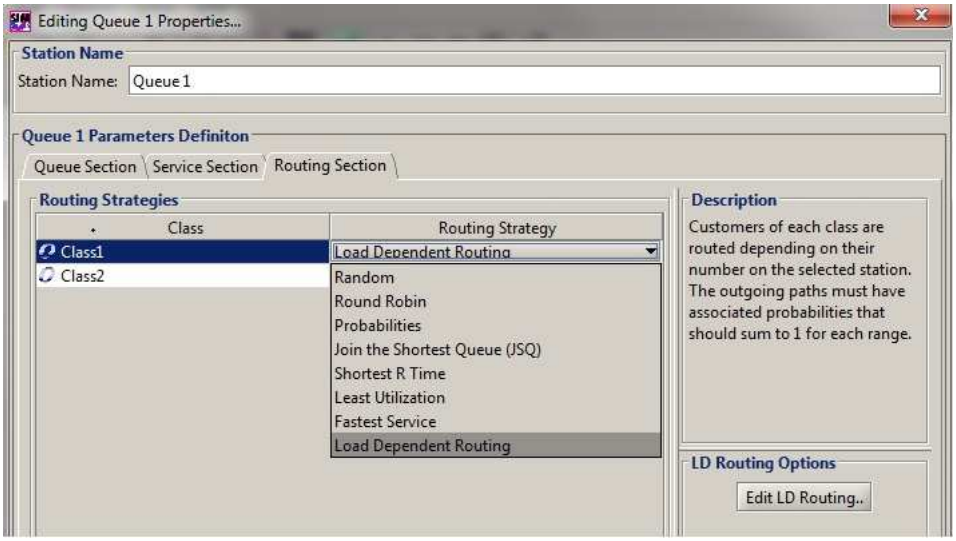


Figure 3.55: Selection of the `Load Dependent Routing` for *Class1* customers.

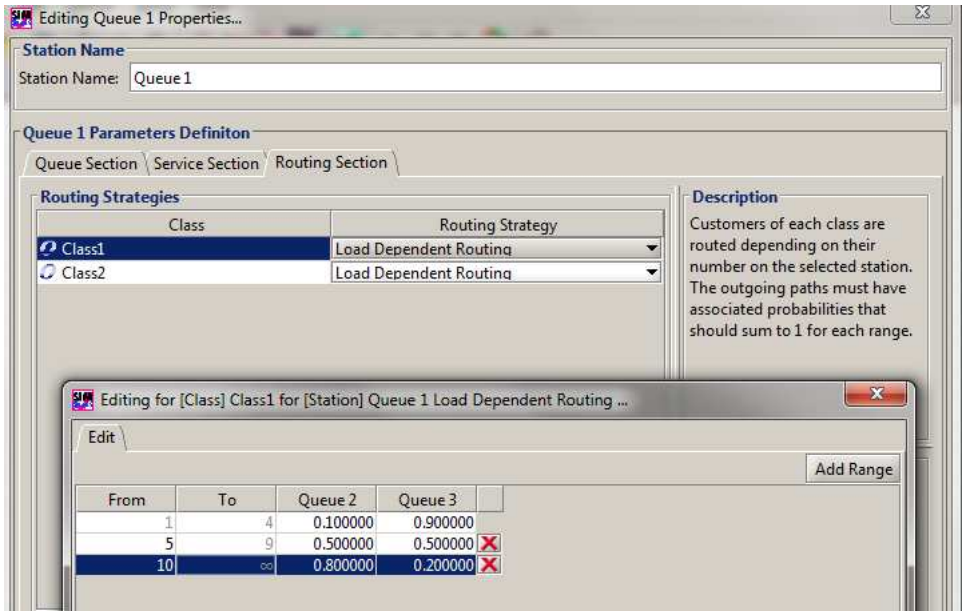


Figure 3.56: `LD Routing`: definition of the routing probabilities for *Class1* customers.

implemented in JSIM is not of the preemptive-resume type.

- **LCFS (Priority):** The next customer to be served is one with the highest priority (conventionally a *small* priority number means low priority), so an arriving customer can only jump ahead of the queue of the other customers with equal or smaller priority. Customers with the same priority are served LCFS.
- **RAND:** An arriving customer joins the queue at a random position and will be served in a random order (*with or without priority* among multiple classes). When the `priorities` of the classes are different, this policy is applied within each priority group separately.
- **SJF (Shortest Job First):** The customer with the *shortest* service time in the queue is the first to be executed (*with or without priority* among multiple classes). When the `priorities` of the classes are different, this policy is applied within each priority group separately.
- **LJF (Longest Job First):** The customer with the *longest* service time in the queue is the first to



Figure 3.57: The `Queueing Station` icon.

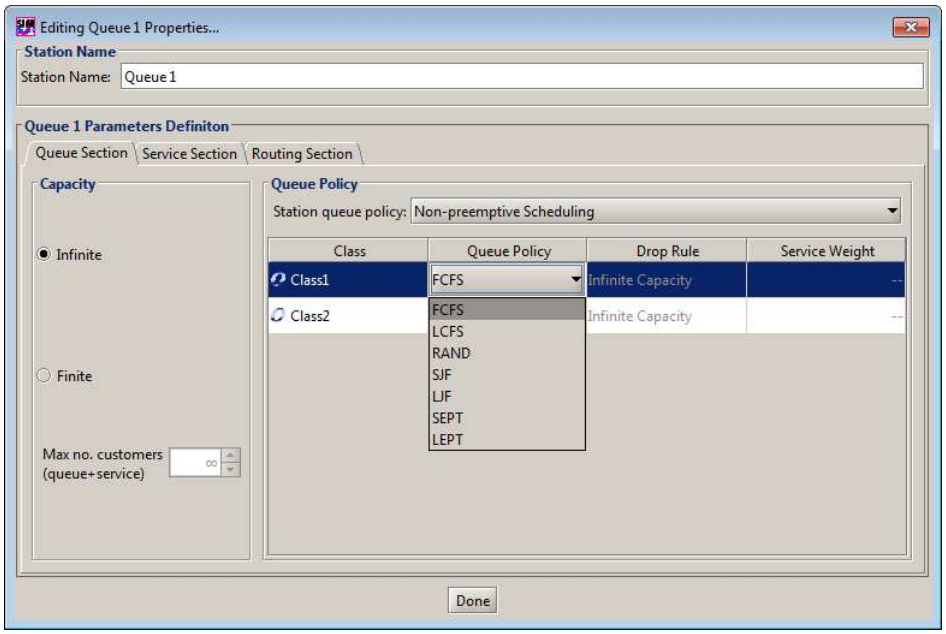


Figure 3.58: Window for editing the `Queueing Station` properties (`Non-preemptive Scheduling`).

be executed (*with or without priority* among multiple classes). When the `priorities` of the classes are different, this policy is applied within each priority group separately.

- **SEPT (Shortest Expected Processing Time):** Customers of the class that has a service time distribution with the *smallest* mean are served first (*with or without priority* among multiple classes). When the `priorities` of the classes are different, this policy is applied within each priority group separately.
- **LEPT (Longest Expected Processing Time):** Customers of the class that has a service time distribution with the *largest* mean are served first (*with or without priority* among multiple classes). When the `priorities` of the classes are different, this policy is applied within each priority group separately.
- **EDD (Earliest Due Date):** The first customer to be executed is the one with the smallest deadline configured at the station. In case of ties, the jobs with identical deadlines are processed in FCFS order.

The available preemptive scheduling disciplines are distinguished in `Processor Sharing` type policies and other `Preemptive Scheduling`.
The `Processor Sharing` type policies (see Fig.3.59) are as follows:

- **PS (Processor Sharing):** The server works as all the customers in the station are processed simultaneously, but devotes to each of them a fraction of its total capacity that is *inversely* proportional to the number of customers. It can be seen as a RR scheduling with a *quantum size* zero so that the control of the server circulates infinitely rapidly among all the customers. If the station has *multiple servers* (greater than 1), when there are more customers in execution than servers, each server splits its capacity equally across the customers. In the other case, the capacity of a server is completely allocated to the customer it has to execute, so its service time is the same as with FCFS, for example. That is, if a multi-server PS queue has m servers but only $n < m$ resident customers, then, each of them will run inside a single server as-if it was in execution alone while leaving $m - n$ servers inactive. Thus, it will never happens that a customer is executed with a service time less than the time it requires as if it is alone in the station.
- **GPS (Generalized Processor Sharing):** Each class k in the station is assigned a positive weight w_k, and the service capacity is divided among all the classes in proportion to their assigned weights. The fraction of the service capacity allotted to a class-k customer is thus given by

$$r_k = \frac{w_k}{n_k \sum_l w_l}$$

where n_k is the number of class-k customers in the station. When no customers of a class are present in the station, the service portion of that class is allocated to other active classes.

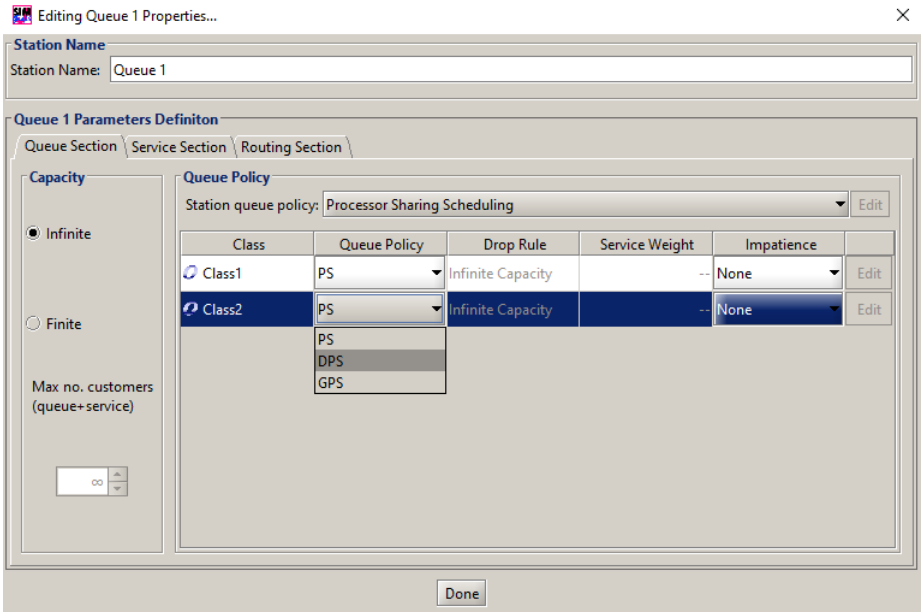


Figure 3.59: Window for editing the `Queueing Station` properties (`Processor Sharing Scheduling`).

– **DPS (Discriminatory Processor Sharing):** Each class-k customer in the station is assigned a positive weight w_k, and the service capacity is shared among all the customers in accordance with their assigned weights. The fraction of the service capacity allotted to a class-k customer can be obtained as

$$r_k = \frac{w_k}{\sum_l w_l n_l}$$

where n_l is the number of class-l customers in the station. Note that unlike GPS, this policy does not guarantee a minimum portion of the service capacity for each class.

For PS, DPS and GPS it is also possible to specify a limit on the maximum number of jobs that can share the server at any given time. Often, such variants of PS are referred to as limited processor sharing. Figure 3.61 shows the dialog window to configure this option, which can be access after clicking the *Edit* button on the right of *Processor Sharing Scheduling* in the Queue Section.
The other `Preemptive Scheduling` type policies are as follows:

– **Last Come First Served Preemptive Resume (LCFS-PR)**: Under this policy, an arriving job goes to the top of the queue and into service immediately. If there is already a job in service, the server pauses the current job, and immediately begin servicing the new job. When a job in service is completed, the last preempted job returns into service, resuming work from where it left.
– **First Come First Served Preemptive Resume (FCFS-PR)**: This policy behaves similarly to LCFS-PR, but only jobs of higher priority can preempt the job in service. Jobs within the same priority class keep being served in FCFS order.
– **Shortest Remaining Processing Time (SRPT)**: This is a famous policy which is known to be optimal in many scheduling scenarios. At any given time, the server schedules for execution the job with the shortest remaining processing time to complete. In case of ties, the job currently in service remains in execution. Under SRPT, it is normally the case that jobs in execution are never preempted until completion unless a new arrival with shortest processing time comes into the queue.
– **EDF (Earliest Deadline First):** The first customer to be executed is the one with the smallest deadline configured at the station. The execution is preempted in case a job with earlier deadline arrives. In case of ties, the jobs with identical deadlines are processed in FCFS order.
– **TBS (Total Bandwidth Server):** The Total Bandwidth Server algorithm is used in real-time scheduling and follows these assumptions:
  1. All periodic jobs $\tau_i : i = 1, ..., n$ have hard deadlines and their schedulability must be guaranteed off line.
  2. All aperiodic jobs $J_i : i = 1, ..., m$ do not have deadlines and must be scheduled as soon as possible, but without jeopardizing the schedulability of the periodic tasks.
  3. Each periodic task τ_i has a period T_i, a computation time C_i, and a relative deadline D_i equal to its period.

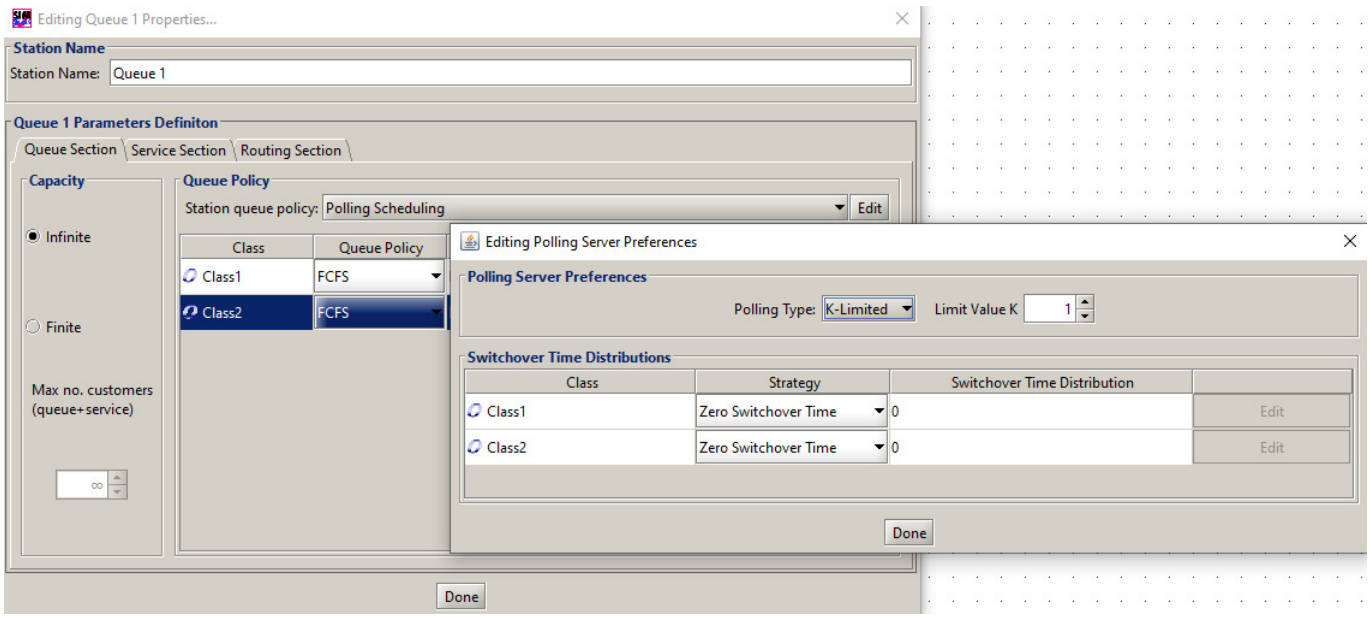


Figure 3.60: Window for editing the `Queueing Station` properties (`Polling Scheduling`).

4. All periodic tasks are simultaneously activated at time t = 0.
5. Each aperiodic task has a known computation time but an unknown arrival time.

In the TBS algorithm, periodic tasks are scheduled using EDF. When an aperiodic job J_k arrives, it gets assigned a deadline using the following equation:

$$d_k = max(r_k, d_{k-1} + p_k/U_s)$$

where:

* d_k : The deadline to be assigned to J_k
* d_{k-1} : The deadline assigned to the previous aperiodic job
* r_k : The time at which J_k arrives
* p_k : The processing time of J_k
* U_s : The server utilisation, i.e., its bandwidth

The idea is that deadlines are assigned such that the overall processor utilisation of the aperiodic tasks does not exceed our server utilisation factor. Once a deadline is assigned, we simply schedule the job via EDF in the same manner as our periodic tasks.
Because JMT does not have concepts of periodic/aperiodic tasks, the following variation of TBS is implemented:

* Job classes with a higher priority than that of the job class which uses TBS are seen as hard "periodic" tasks. The reasoning behind this is that the idea of having a hard deadline implies that that job must be scheduled to meet its deadline, i.e., it must be prioritised over other jobs.
* The server utilization factor U_s is recalculated each time a deadline is to be assigned by TBS. It is calculated as $1 - U_p$, where U_p is a stochastic estimate of the summed processor utilisation of all periodic jobs at that point in time.

The Queue Section allows to specify for each class an *Impatience* policy, which is a mechanism to describe the abandonment of customers that have waited too long in queue. Two forms of impatiences are presently supported:

– **Balking**. A job is said to have balked when it leaves the entire system without ever joining the queue. Balking can only be applied to open systems, but not closed systems. A job cannot balk if the queue is empty. Every job class will have its own probability distribution that determines the balking probability of the job. The probability that the job balks before entering the queue is determined by the user, and is itself designed to be a function of the queue length, specified in a

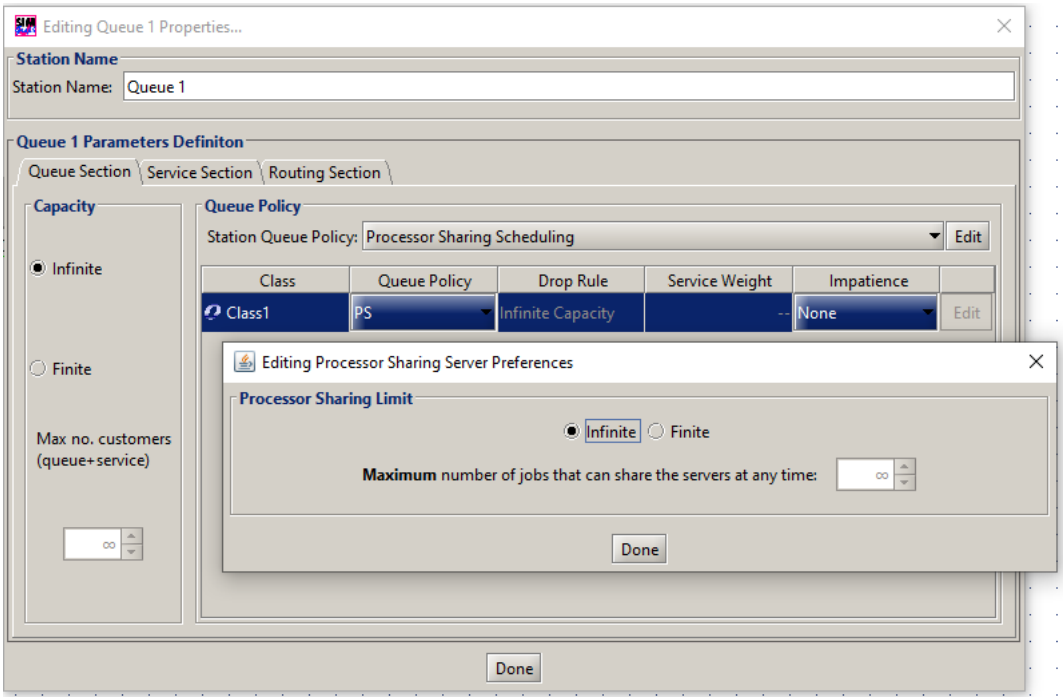


Figure 3.61: Window for editing the limited processor sharing configuration in the Queue Section.

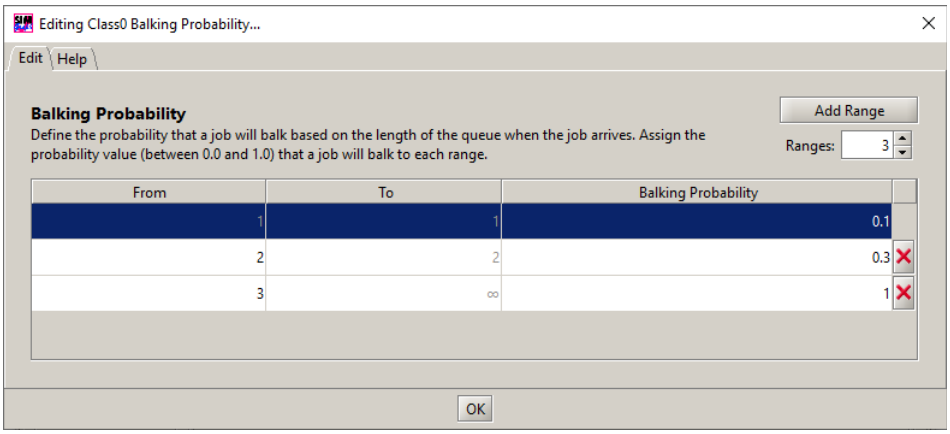


Figure 3.62: Window for editing the `Balking` properties in the Queue Section.

dedicated dialog window shown in Fig.3.62. When a job balks, the information will be recorded using a new performance index called *Balking Rate*.

– **Reneging**. This form is impatience is also called abandonment: a job is said to have reneged when it enters the queue but abandons it at some point in the future without ever receiving service. Reneging can only be applied to open systems, but not closed systems. The period where a reneging can occur for a job starts when the job joins a queue and ends when it starts receiving service, therefore it does not apply to queues with policies like Processor Sharing or LCFS-PR since arriving jobs are immediately served. The probability distribution of the reneging time is specified by the user, and the actual time at which the job reneges is determined by JMT the moment the job enters the queue. When a job reneges, it is dropped entirely from the system and is not routed anywhere else. When a job reneges, the information will be recorded using a new performance index called *Reneging Rate*.

JMT makes available also `Polling Scheduling` disciplines. A polling system is one where a group of servers serves multiple buffers. In JMT, a buffer represents a waiting space within a single queue where jobs of the same class wait for admission to the server. That is, polling in JMT is performed within the same queue but alternating service of jobs of different classes.
The queue consists of R buffers, one for each class, typically infinite in size. The server serves jobs from one buffer at a time in an ordered cycle. The term "polling" comes as a result of the server polling each buffer for jobs. If the station has multiple servers, they move to the next buffer together when the

processing of the current buffer is completed.
As shown in Fig.3.60, each buffer can be specified with its own scheduling policy (e.g., FCFS) that the polling servers will use to obtain the next job to process. A separate dialog window can be reached via the "Edit" button next to "Polling Scheduling" to specify the parameters of the polling policies, such as the K value of K-Limited polling described below.
In polling systems, switching the queue in service will incur a switchover time period. This represents the cost in terms of time for the changes in service. In some systems this cost could be zero. Crucially, a high switchover period punishes frequent switching. The same dialog window shown in Fig.3.60 is used in JMT to specify the switchover time distribution for a server that moves on leaving behind the buffer for the specified class. This time can be either deterministic or random. For example, indicating a Zero Service Time for *Class1* means that upon completing service of the *Class1* buffer the server will incur no cost to switch to the next buffer for *Class2*. However, if one selects Exponential the switchover time period will be a random number drawn from this distribution.
Warning: JMT presently implements only *wait-and-see* polling scheduling. Namely, if no jobs are present in any of the buffers, then the server stops until a job arrives. This differs from variants where the server keeps cycling in the case where there is a switchover time period.
The following polling types are presently supported in JMT:

- **Gated Polling.** A gated polling system services all jobs currently in the current buffer whenever the server visits. This means that if buffer r has n_r jobs when the server visits, all and only these n_r jobs will receive service, before the server moves on to the next buffer.
- **Exhaustive Polling.** The server in an exhaustive polling system will only move on from a buffer once it is empty. It will continue serving even the jobs that arrive after the start of the service cycle for the current buffer.
- **K-Limited Polling.** In the K-limited polling system, the server will serve K jobs from one buffer and then switch to the next buffer, and so on. If the buffer empties before K jobs have been served, the server will still move on.

Service Section
The service time distribution and the service configuration, in particular the number of servers (*default* is 1), should be defined. The load *dependent*, *independent* or *zero service time* characteristic of the service times must be specified for each class of customers through the menus of Figure 3.67 and Figure 3.68.
For a basic description, please refer to *Service Section* of subsection 3.11.4 *Delay Station*.

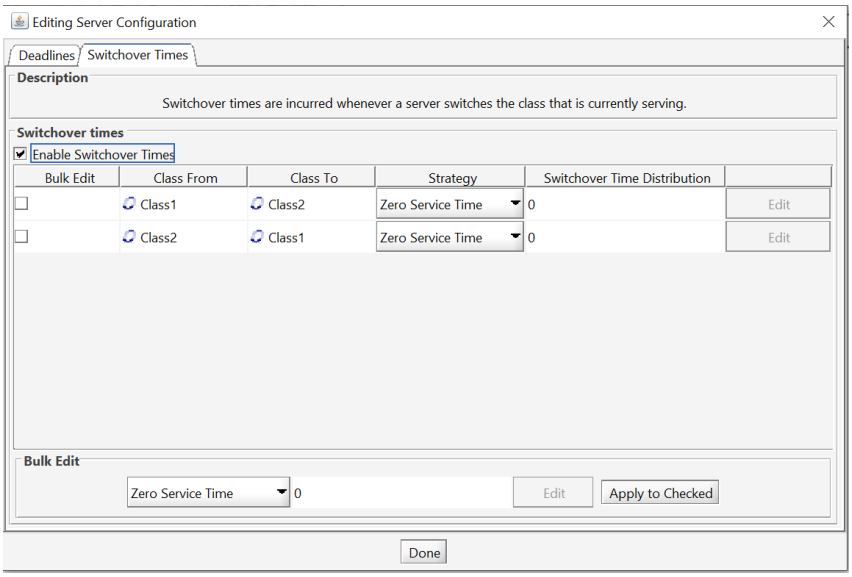


Figure 3.63: Window for editing the `Switchover time` distributions in the Service Section.

- **Switchover times**. For queueing stations alone, the *Service Section* offers an *Edit* button to modify the advanced features. This opens a new dialog where the use can specify the Switchover Times of the model. Clicking on "Enable Switchover Times" enables to specify a distribution of time for the delay that is incurred whenever one of the servers switches from processing a job of a class to a job of another class. A figure illustrating the switchover time dialog panel is shown in Figure 3.63.

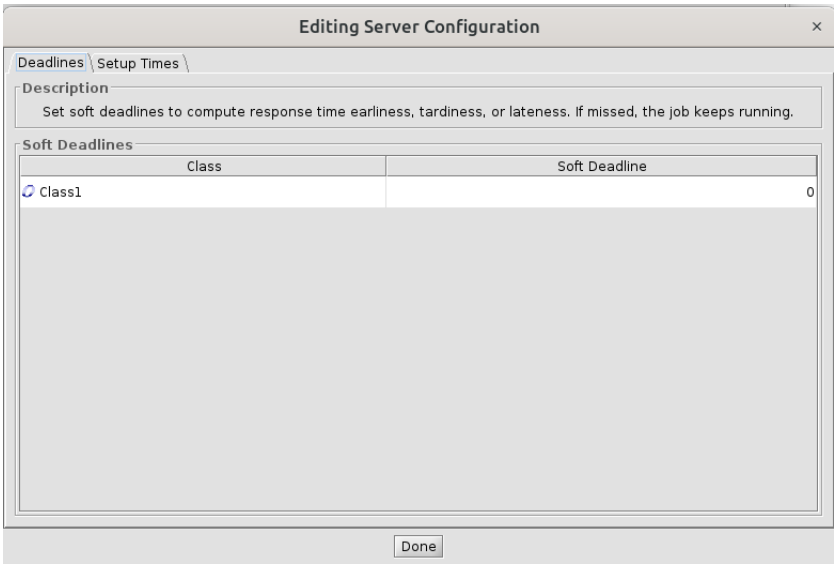


Figure 3.64: Window for editing the `Soft Deadlines` in the Service Section.

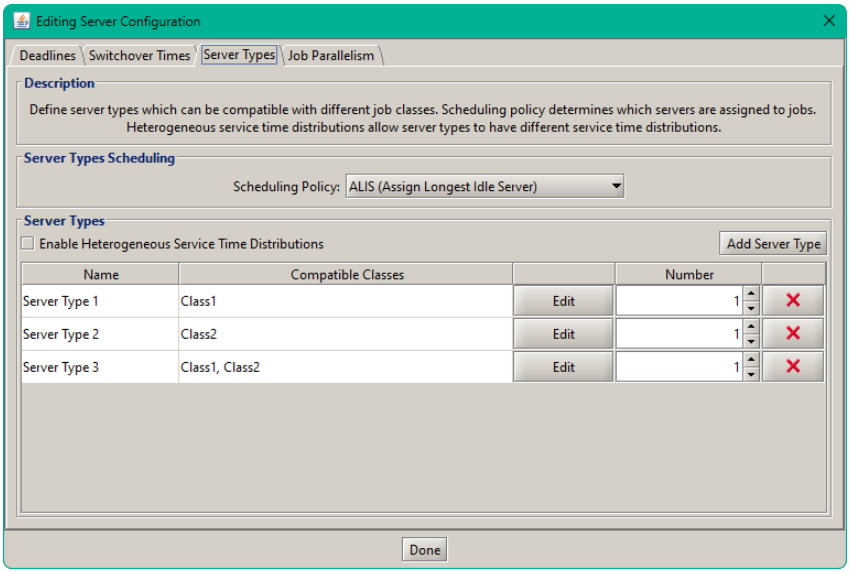


Figure 3.65: Window for editing the `Server Types` in the Service Section.

The switchover time therefore will be added to the service time of that server for the new class, however it will not produce useful work. It is possible to measure only the useful part of the server utilization by means of the *Effective Utilization* performance index.

– **Station deadlines**. In the same window of the Setup Times it is possible to choose the *Deadlines* tab to enter the (relative) *soft deadlines* associated to each class of jobs at the station. Such deadlines are computed from the time the job arrives to the station and refer to the response time of a single visit. If the job missed the deadline, it will still continue running until completion. A figure illustrating the deadline dialog panel is shown in Figure 3.64. Soft deadlines are also available in delay stations.

– **Server types**. This tab, shown in Figure 3.65, handles server compatibilities. This is the concept of having different server types which are each only able to serve certain types of jobs, as specified by a subset of compatible classes. The server can also specify whether multiple servers are available to process a given compatible job. Alongside the implementation of server compatibilities are novel scheduling strategies for assigning servers to jobs, namely:

  * *ALIS: Assign Longest Idle Server.* The most commonly found strategy in server compatibilities research. ALIS puts servers in a queue and assigns idle, compatible servers in the same way as FCFS assigns jobs.
  * *ALFS (Assign Least Flexible Server).* In this context, flexibility of a server refers to the number of compatible job classes. The idea behind this policy is to prioritise less flexible when possible

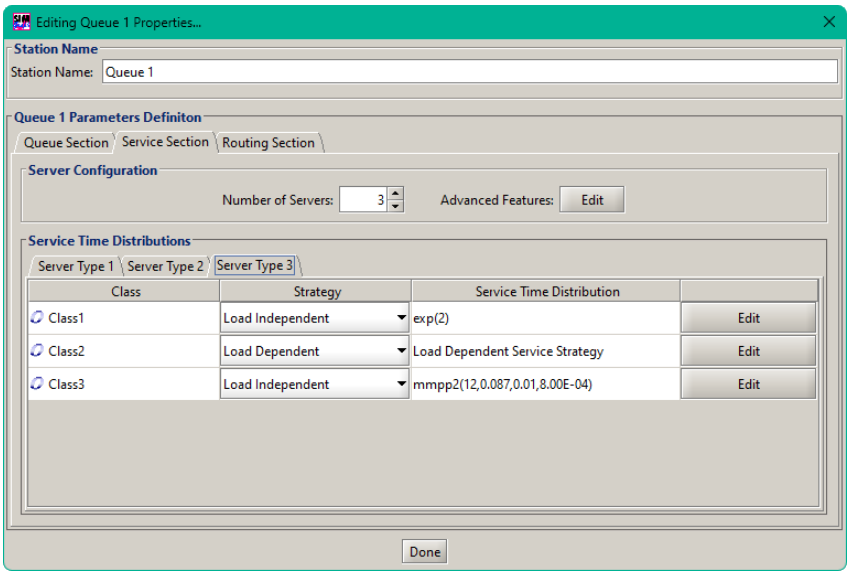


Figure 3.66: Modified of the Service Section after enabling the `Heterogeneous Service Times`.

in order to maximise availability of the more flexible servers which can handle a larger variety of jobs. This strategy minimises situations where there are waiting jobs and idle servers but none of the servers are compatible with any of the waiting jobs.

  * *Fairness.* A modification of ALIS where servers are grouped together by server type when taking into account their idle time.
  * *FSF (Fastest Servers First).* In this strategy, jobs are assigned to the idle, compatible server which is expected to give the shortest service time. This scheduling strategy inspired the addition of heterogeneous servers to the project, since otherwise every server type has the same service time distribution. In order to execute this scheduling policy in JMT, the simulator checks the expected service time for each available server with the given job and assigns the job to the server with the shortest expected service time.
  * *RAIS: Random Assignment to Idle Servers.* When there are multiple compatible, idle servers, jobs are assigned to a random choice.
  * *Order.* This assigns servers based on the order in the user-defined server types table.

- **Heterogeneous Service Time Distributions** Note that, with the introduction of server compatibilities, we now have systems where servers within the same node are not the same. Considering this, it makes sense that they should not be restricted to have identical service time distributions. By enabling this option in the Server Types tab, it is indeed possible to assign back in the Service Section different service distributions to different servers, as show in Figure 3.66.
- **Job Parallelism**. This tab allows to specify the number of servers required simultaneously by a job for its processing. This value can be differentiated across job classes. For example, if a waiting job requires a parallelism of 2 but a single server is available, then the job is prevented from starting. This feature and server types are compatible with each other, allowing to specify fairly advanced models.

Routing Section

In the routing section, for each class of customers, you can decide the path followed by the completed jobs are routed to the devices connected to station for which the routing strategy is defined. For each class, the user should select the algorithm you want to use in order to determine the outgoing connection followed by a customer. The following algorithms are available:
- Random
- Round Robin (with and without weights)
- Probabilities
- Join the Shortest Queue (JSQ)
- Shortest Response Time
- Least Utilization
- Fastest Service
- Load Dependent Routing

- Power of k choices (with and without memory)
- Class Switch
- Disabled.

To know details about these algorithms, please refer to the `Routing Section` of subsection 3.11.1-*Source Station*.

3.11.4 Delay Station

Customers that arrive at this station are delayed for the amount of time that defines the station service time. They do not experience any queuing, since a delay station is modelled as a station with an infinite number of servers with the same service time. Thus response time of delay stations is equal to the service time, $R_i = S_i$. Furthermore, the queue length corresponds in this case to the number of customers receiving service since there is no waiting: $N_i = R_i X_i = S_i X_i = U_i$. *In delay stations*, for consistency with Little's law, the *utilization* is computed as the *average number of customers* in the station, and thus it may be *greater than 1*!
Delay stations are used when it is necessary to produce some known average delay. Application of delay stations may be to model transmission time (download, upload, ...) of large amounts of data over a network (Internet, lans, ...) or to model users think time at the browsers.

Setting or changing the properties
Double click on the icon representing the delay station to see the property panel.

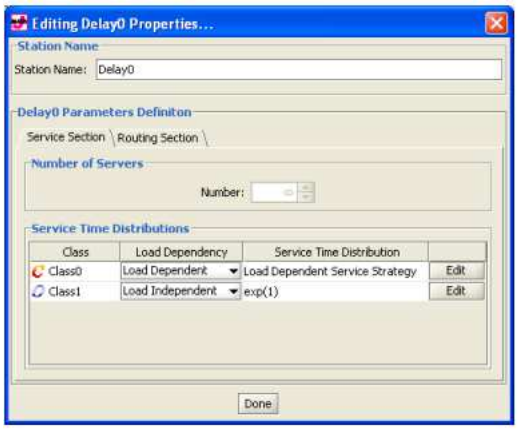


Figure 3.67: Windows for the editing of `delay properties`.

Service Section
Delay stations are infinite servers with the same service time characteristics; the service time distribution should be defined. The infinite numbers of servers provide the same average response time for all jobs, as no job waits in queue for service. The load *dependent* or *independent* characteristic of the service time must be specified for each class of customers through the menus of Figure 3.67 and Figure 3.68.

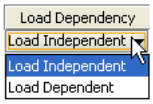


Figure 3.68: Selecting load dependency.

- **Load Independent**: A load independent service indicates that, regardless of the number of customers that are in the station, the system will serve all the customers following a fixed policy modelled by the chosen statistical distribution. To choose the Distribution press the `Edit` button and insert all the parameters from the window of Figure 3.69.
  The following service time distributions (see section 3.6) are supported: *Burst (General), Burst (MAP), Burst (MMPP2), Coxian, Deterministic, Erlang, Exponential, Gamma, Hyperexponential, Normal, Pareto, Phase-Type, Replayer, Uniform.*
- **Load Dependent**: A load dependent service time indicates that the amount of time the server spends with each customer depends upon the current number of customers in the station. A set of

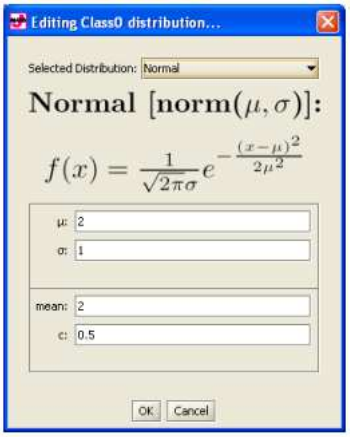


Figure 3.69: Editing of the parameters of a `distribution`.

intervals for the number for jobs in the station is specified, either by adding one range at a time via the `Add Range` button or by specifying the total number at once. For each range its lower (`From` button) value must be specified. Automatically its upper value (`To` field) is computed. Each range of values can be associated with different service times, distribution, mean and coefficient of variation, or a subset of them. To set the parameters of a Load Dependent service time, click the `Edit` button to edit the Service Time Distribution. The panel of Figure 3.70 appears, then specify the parameters for each range.
For each range of the number of customers the following parameters must be described:

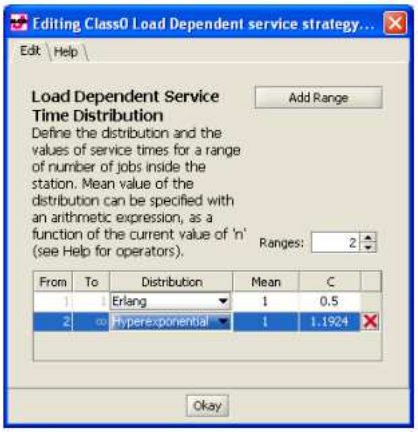


Figure 3.70: Window for editing a `load dependent` service strategy.

- **Distribution**: you can choose among Coxian, Deterministic, Erlang, Exponential, Gamma, Hyperexponential, Normal, Pareto, Uniform distributions.
- **Mean**: the mean value of each distribution is specified in the "Mean" form by double clicking on it. Insert a number or an arithmetic expression that will be evaluated with JFEP - Java Fast Expression Parser. For a complete list of the command supported by JFEP you can read the `Help` tab or see the JFEP web site at http://jfep.sourceforge.net/
- **C**: the coefficient of variation of each distribution (when c exists) can be specified by double clicking on the c form. For example, in Figure 3.70 two distributions are defined: when there is only one customer in the station the service times are generated according to an Erlang distribution with *mean=1* and *c=0.5*. For any number of customers greater than 1 in the station, the service times are generated according to an Hyperexponential distribution with *mean=1* and $c = 1.19$.

To delete a range click on the *delete* icon ✖ at the end of the correspondent row.

– **Zero Service Time:** A `Zero Service Time` strategy is used to model stations having negligible (or null) service time. A station with *zero service time* may have a queue if the output customers are routed to a station with a `BAS blocking` scheduler.

Routing Section

In the routing section, for each class of customers, you can decide how the completed jobs are routed to the other devices connected to station for which the routing strategy is defined. For each class, the



Figure 3.71: Selecting the `Routing Strategy`.

user should select the algorithm you want to use for outgoing connections. The following algorithms are available:
- Random
- Round Robin (with and without weights)
- Probabilities
- Join the Shortest Queue (JSQ)
- Shortest Response time
- Least Utilization
- Fastest Service
- Load Dependent Routing
- Power of k choices (with and without memory)
- Disabled.

To know details about these algorithms, please refer to subsection 3.11.1 of *Source Station*.

3.11.5 Fork Station

A `Fork` station splits the jobs into several *tasks* that are executed in parallel. No service time is allocated for this operation, thus there is no service time specification request. Tasks are routed on the Fork outgoing links (Figure 3.72).

Unlike classical queueing theory Fork/Join structures, a JSIM `Fork` station is not associated with a cor-

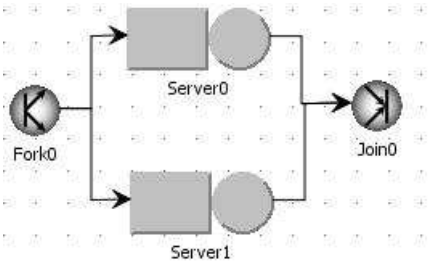


Figure 3.72: `Fork` and `Join` stations.

respondent `Join` station automatically. Any combination of queueing stations, Finite Capacity Regions, fork-join, routing stations, loggers, etc., is possible after a Fork station. This feature allows the modeling of *very general* models of parallel behaviors, of which the traditional Fork/Join structure is a special case. In JSIM, the *classical* `Fork/Join` queue structure, i.e. the most simple one referred to as `Standard Fork`, is obtained by connecting the `Fork` station to as many queue stations as the degree of parallelism requested by the user, generating one or more tasks per outgoing link (as in Figure 3.72). Each queue station is then connected to a `Join` station, where the executions of the tasks are synchronized and the job is reconstructed.

The `Fork/Join` structure implemented in the JSIM is very powerful. In addition to the `Standard` strategies there are *four advanced fork* strategies and *two advanced join* strategies. Each class of customers has *its own* strategy *independently* of those adopted by the other classes.

Set or change of properties

The `Fork` station consists of *three* sections: `Capacity Section, Queue Section` and `Fork Strategies` (see Fig.3.73).

– **Capacity Section:**
The user may define the maximum number of customers (jobs) that can be executed in a Fork-Join section simultaneously. Clearly, it makes sense only if the user define a corresponding `Join` station

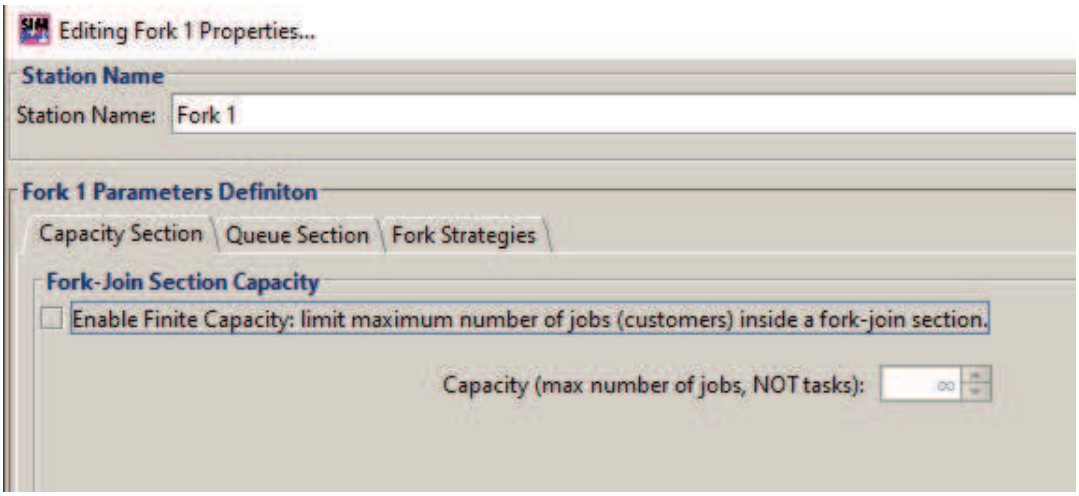


Figure 3.73: Initial window of the editing of the `Fork` station properties

.

matching the Fork one. It can be *finite*, in this case when the limit is reached the jobs wait in the queue of the Fork station. A job is removed from the queue and served (i.e., split into *tasks*), then it is reconstructed at the corresponding `Join` station and leaves it. Capacity is defined checking the `Enable Finite capacity` box in the window of Fig.3.73. The *default* value is infinite.

– **Queue Section**
Please refer to `Queue Section` of subsection 3.11.3 `Queueing Station`

– **Fork Strategies**:
In this section the user may select a `Fork strategy` among the following strategies: `Standard, Branch Probabilities, Random Subset, Class Switch,` and `Multi-Branch Class Switch` (see Fig.3.74).

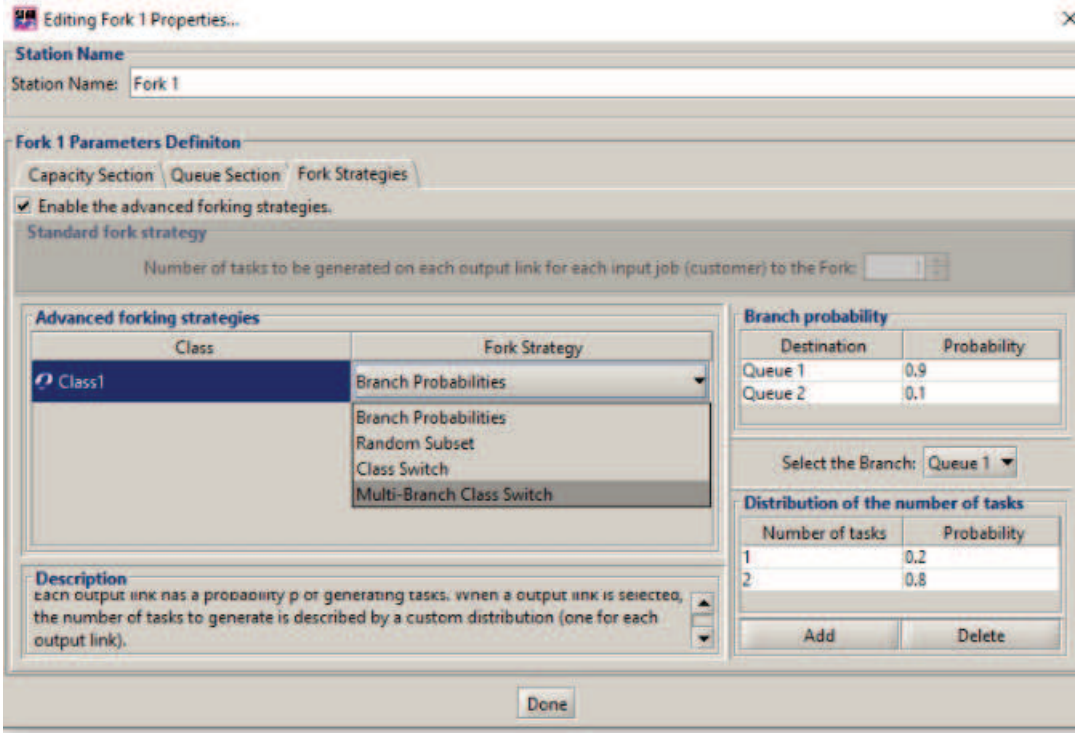


Figure 3.74: Selection of a `Fork Strategy` (in this case `Branch Probabilities`)

.

1. **Standard Strategy**:
This is the default strategy when the box `Enable the Advanced Forking Strategies` is NOT checked. In this case the number of tasks created in each outgoing link (referred to as *forking*

degree) for each job arriving at the `Fork` station is the same. By *default* the forking degree is 1, but it can be modified through a counter (see Fig.3.74)

2. **Branch Probabilities**:
   The probability p_k that the $k-$th branch (path) receives forked tasks must be specified. For example, in Fig.3.74 the branch between `Fork1` and `Queue1` stations has 90% probability to receive generated tasks while the one between `Fork1` and `Queue2` has only 10% of probability. Let us remark that to increase the generality of strategies that may be modeled, the sum of the probabilities of all the branches may be different from 1. Thus, it may also be simulated cases in which a job has no generated tasks or in which a branch always receives tasks.
   For each selected branch (e.g., `Queue1` in Fig.3.74), the *distribution* of the number of tasks that will be routed can be described. Clearly, the sum of these probabilities must be 1. The `Add` and `Delete` buttons may be used to manage the elements in the probability table.
3. **Random Subset**:
   In this strategy, a user can specify the `Probability` p_n that the `Fork` generates the tasks on n branches, i.e., the probability that n output branches receive the forked tasks (see `No. of branches` in Fig.3.75). One task is generated per output branch considered and selected randomly according to the probability values defined by the user. Thus, the number of tasks is not known in advance. For example, according to the random policy specified for `Class1`, in Fig.3.75 for 80% of times only one branch is selected to route the single task generated.

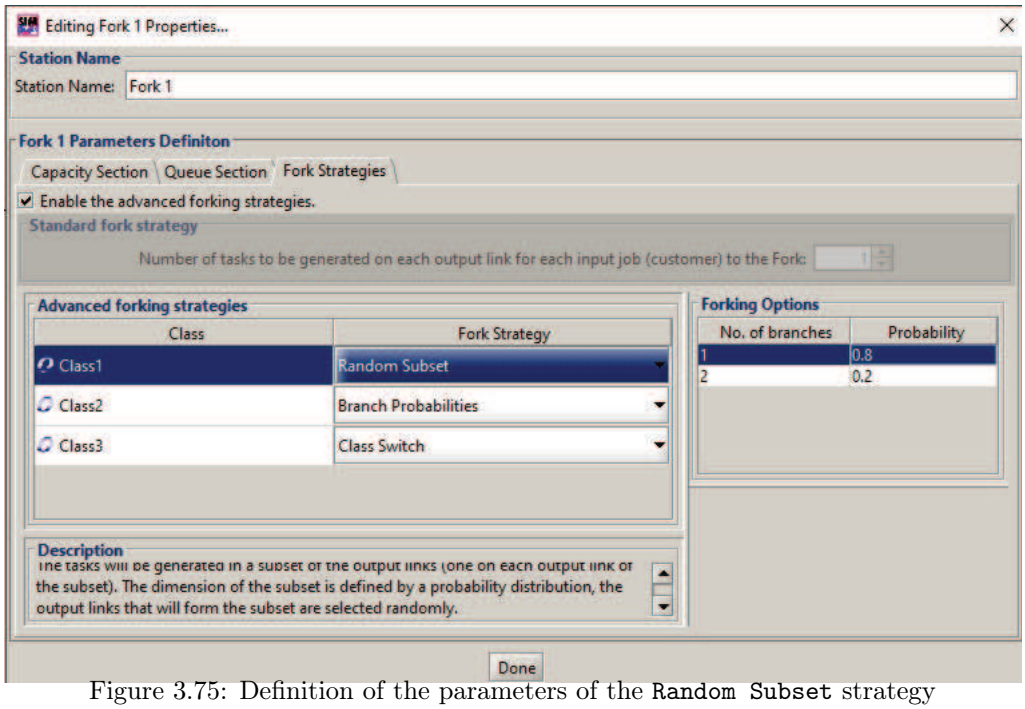


Figure 3.75: Definition of the parameters of the `Random Subset` strategy

.

4. **Class Switch**:
   This policy allows the generation of tasks with different characteristics, i.e., of *different classes* for each job. A user may specify for each `Class` of jobs, the number of tasks of the various classes defined in the model that are generated and routed on each branch (the *same mix* on each branch). For example, based on the parameters defined in Fig.3.76, for each `Class1` job arriving at the `Fork`, *12 tasks* (1 of `Class1`, 9 of `Class2`, and 2 of `Class3`, respectively) are generated *on each branch*. Let us point out that the class of the job that will be released by the Join will be in any case the *same* of the *father job* (the same it has when arrives at the Fork), independently of the classes generated inside the Fork by the `Class Switch` strategy.
5. **Multi-Branch Class Switch**:
   This strategy generalizes the `Class Switch` one. It allows to define a mix of classes of tasks to be generated that may be different for each branch. For example, based on the parameters defined in Fig.3.77, for each `Class1` job arriving at the `Fork` on the branch towards `Queue1`, 18 tasks will be routed (10 of `Class1`, 5 of `Class2`, and 3 of `Class3`, respectively). The mix of tasks of the various classes *may be different* for the branch towards `Queue2`.

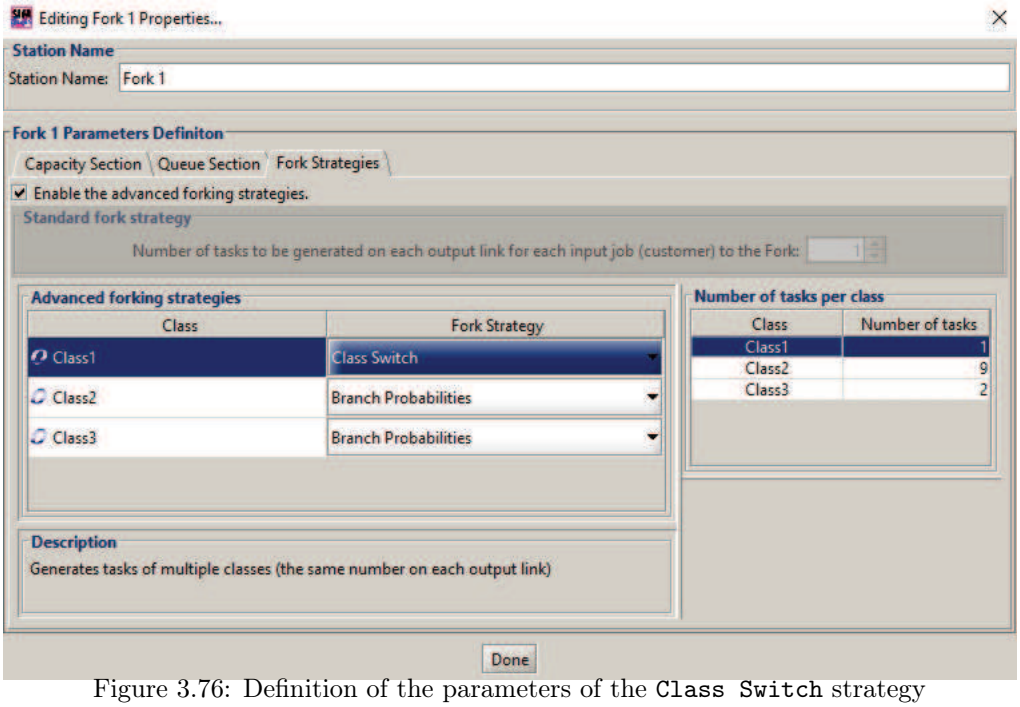


Figure 3.76: Definition of the parameters of the `Class Switch` strategy

.

3.11.6 Join Station

`Join` stations are complementary to `Fork` stations (see Figure 3.72). While in classical queueing network theory, tasks arrive at a `Join` station only from the corresponding `Fork` station, in JSIMgraph, a `Join` station may have incoming links from stations other than the corresponding `Fork` one. This to increase the generality of the applications that can be modeled.
A `Join` station has *no service time*, as it is only used to recombine the tasks a job had been previously split into and then route the job to the following station(s). When a task arrives at a `Join` station, it waits until all (or *other combinations* depending on the `Join strategy` adopted) its sibling tasks have arrived. At this time, the original job is recomposed and routed to the next station. If a job arrives from a station other than the corresponding `Fork`, e.g., a job that was not split, it is immediately routed to the next station. In this case the `Join` station operates as a routing station.
The Join station of JSIM is very powerful. In addition to the `Standard Strategy` (*all* the tasks are waited at the `Join` before to release the job) two other *advanced join strategies* are available: `Quorum` and `Guard` (see Fig.3.78).

Set or change of properties
The `Join` station consists of *two* sections: `Routing Section` and `Join Strategy`.

- **Routing Section:**
  In the `Routing Section`, for each class of jobs defined in the model, a user may define a different routing path of the jobs that exit the Join station (see Fig.3.79). The following algorithms are available:
  - Random
  - Round Robin (with and without weights)
  - Probabilities
  - Join the Shortest Queue (JSQ)
  - Shortest Response time
  - Least Utilization
  - Fastest Service
  - Load dependent Routing
  - Power of k choices (with and without memory)
  - Disabled.

  To know details about these algorithms, please refer to the subsection 3.11.1 *Source Station*.

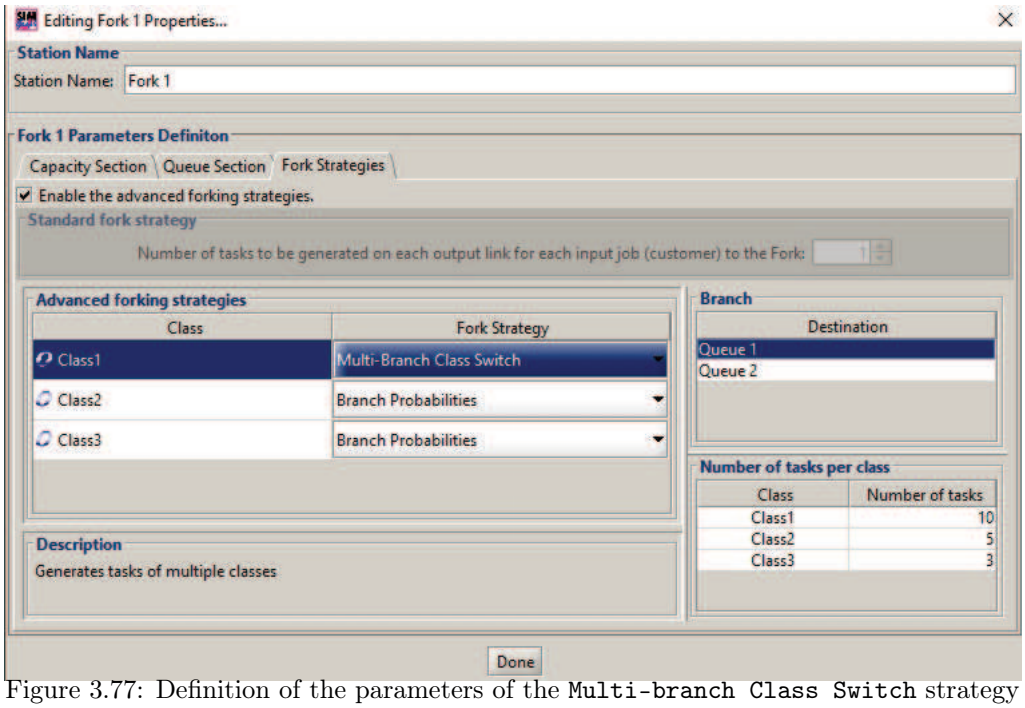


Figure 3.77: Definition of the parameters of the `Multi-branch Class Switch` strategy

.

- **Join Strategy**

  1. **Standard Join**
     This is the *usual* policy adopted by `Join` stations. *All* the tasks of a job generated by the `Fork` must arrive at the `Join` before that the job will be released. Thus no parameters are requested (see Fig.3.78).
  2. **Quorum**
     With this strategy, when the `Join` has received the number of tasks of a job that the user has specified then it will release the job (e.g., three in Fig.3.80). The number of tasks to be waited at the `Join` can be less than the ones generated by the `Fork`. These tasks may be also of different classes if generated with a `Class Switch` strategy at the `Fork`, but must have the same father job.
  3. **Guard**
     With this strategy, the user may specify per *each class* of jobs the minimum number of tasks to be waited at the `Join` before release the job. This `Guard` can be different for *each class* of jobs. For example, according to the parameters defined in Fig.3.81, 8 Class1, 3 Class2, and 1 Class3 tasks must be arrived at the Join before that the father job of Class 1 will be released. The tasks of different classes must be generated by a `Class Switch` strategy at the Fork and thus have all the same father job.

3.11.7 Routing Station

A *routing* station is a dummy station, with service time equal 0, which is used to create more sophisticated routing strategies by sending customers through one or more such stations. For example, if we want that two thirds of the incoming traffic at station Z to be randomly routed to either station A or B or the remaining third to go either to station C or D, depending on the shortest queue at the two stations, we could implement the following. Add a routing station, Y, to the output of Z station and define random routing for the three stations connected in output (namely, A, B, Y). Then connect Y to C and D and define shortest queue routing JSQ for C and D.

Set or change of properties
With a double click on the routing station icon, the `Editing Routing Station Properties` window is shown. It has only one tab `Routing Section`.

Routing Section:
In the routing section, for each class, you can decide how the completed jobs are routed to the other devices

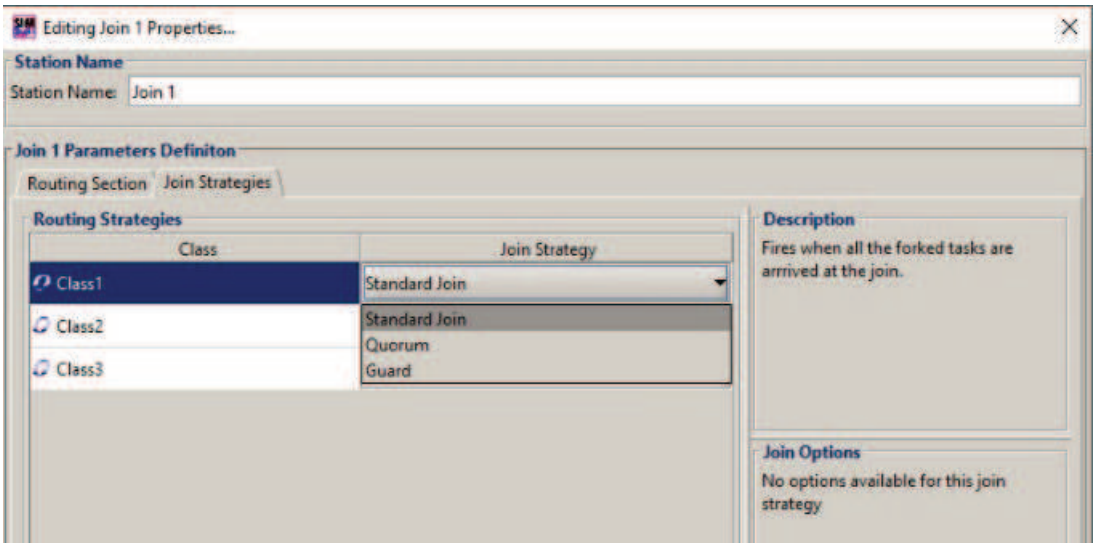


Figure 3.78: Selection of the `Standard Join Strategy` for `Class1` jobs.

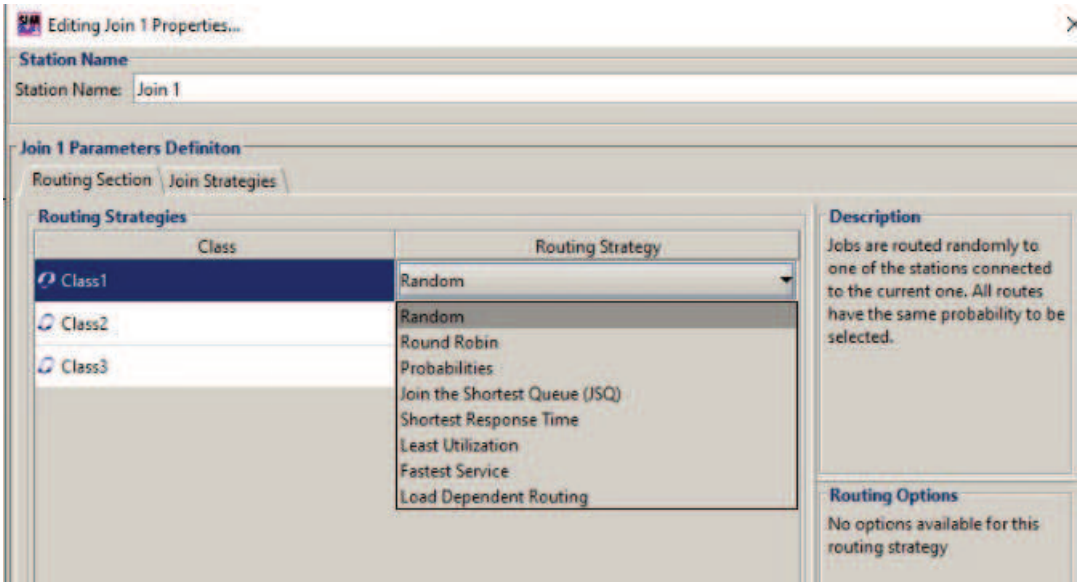


Figure 3.79: Definition of the `Routing` path for `Class1` jobs that are released from the `Join1` station.

connected to the considered station. For each class, the user should select the algorithm applied to determine the path followed by the outgoing customers. The following algorithms are available:
- Random
- Round Robin (with and without weights)
- Probabilities
- Join the Shortest Queue (JSQ)
- Shortest Response time
- Least Utilization
- Fastest Service
- Load Dependent Routing
- Power of k choices (with and without memory)
- Disabled.

To know details about these algorithms, please refer to the subsection 3.11.1 *Source Station*.

3.11.8 Logger Station

A logging station (i.e., *logger*) reads information flowing through it and writes it to a file. In the simplest way, it is a tool to understand and debug the traffic flow moving through the interesting part(s) of the model.

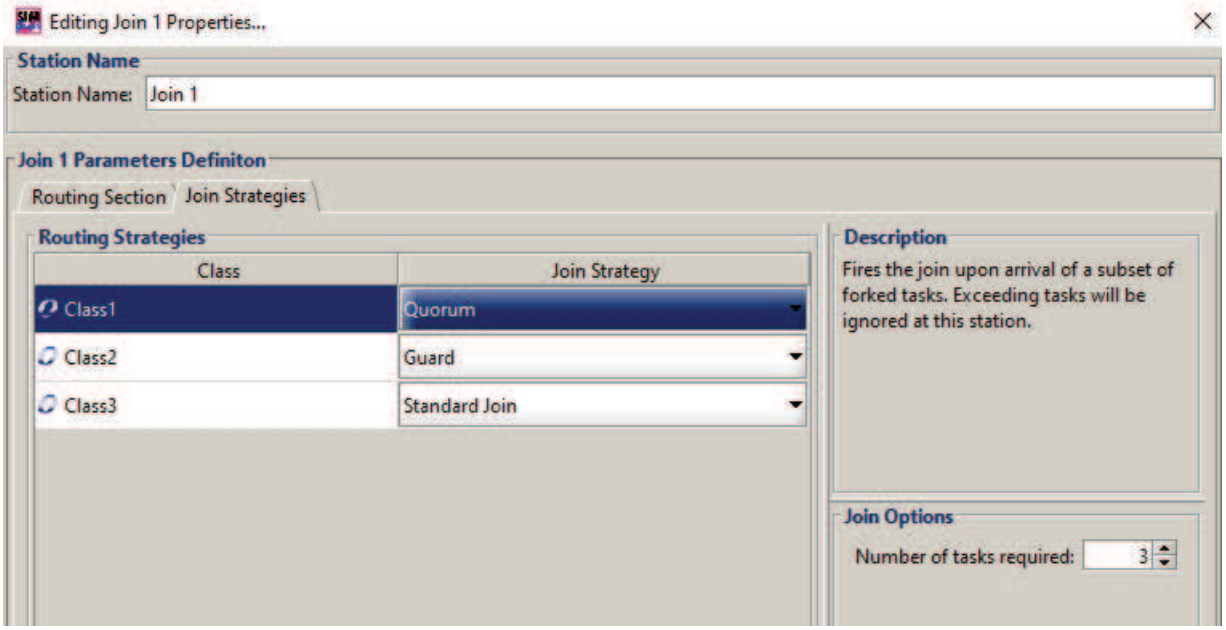


Figure 3.80: Editing of the `Quorum Join Strategy`.

Place the logger station into the model, choose the parameters, and trace the data as it passes through the model.

Set or change of the properties

Double click on the `Logger Station` icon to open the `Editing Logger Properties` station's configuration dialog (see Figure 3.83). There are two sections: `Logger Section` and `Routing Section`.

Logger Section

The configuration properties of the `Logger Section` are:

- **Logging Options:**
  - The Logfile's name to use, either individual or merged (Logger0.csv or global.csv, respectively).
  - The information to log (fields): Logger Name, Job Arrival Time (simulation units), unique Job ID, defined Class ID, Interarrival time of same class, Interarrival time of any class. These fields are found in the next section.
- **Logfile Options:**
  - Browse button: allows changing the directory where logfiles are stored.
  - Overwrite method to use when the logfile exists, and a new simulation needs to overwrite the file. Replace overwrites the file, append adds data to the end of the file.
  - Delimiter chooses the character to separate between.
- **Status** displays the path and file-status of the logfile that is going to be written.

As messages flow through the Logger from one station to another, the following information can be logged:

- `Logger Name`, the name of the logger where the message occured (e.g., *Logger0*)
- `Job arrival time`, this timestamp marks the current simulation time from start of simulation (not seconds)
- `Job ID`, the auto-generated unique sequence number of the message (e.g. 1,2,3...).
- `Class ID`, the name of *customer class* of message (e.g., *Class0*)
- `Interarrival time (same class)`, is a time difference between customer of the same class that passes through the logger
- `Interarrival time (any class)`, is a time difference between customers of any class that passes through the logger.

Once a *Logger* is configured, running a simulation produces a logfile with the chosen fields. An example of logfile output is:

```
LOGGERNAME;TIMESTAMP;JOBID;JOBCLASS;TIMEELAPSED_SAMECLASS; TIMEELAPSED_ANYCLASS
```

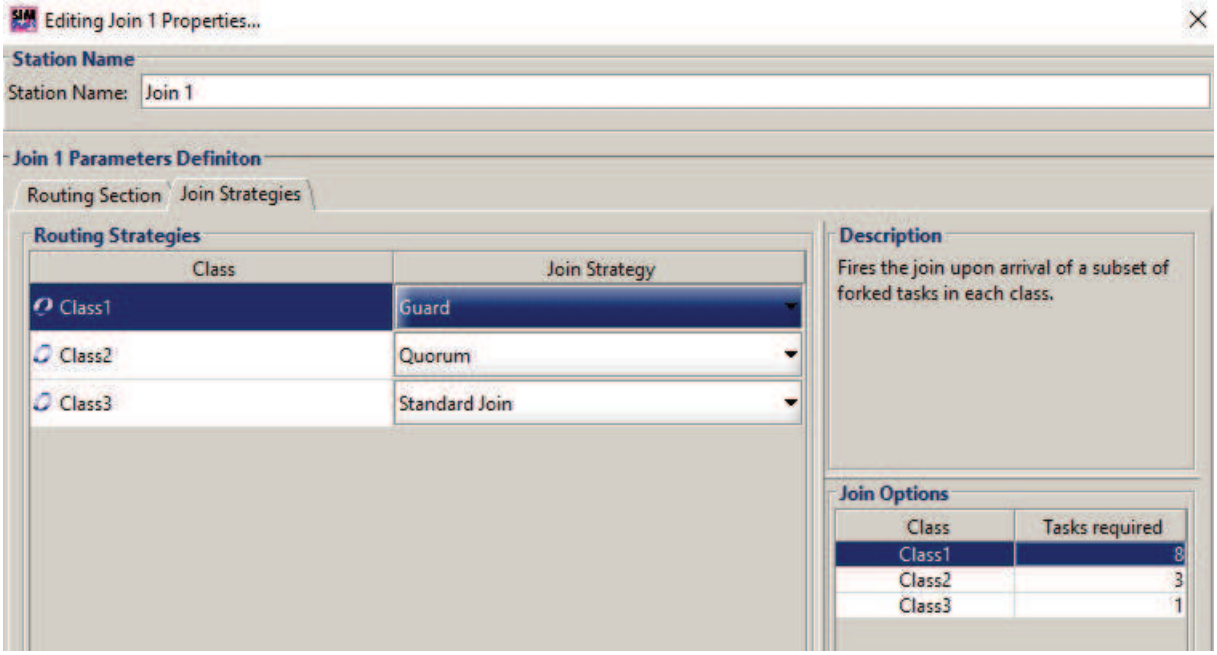


Figure 3.81: Editing of the `Guard Join Strategy`.

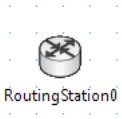


Figure 3.82: The `Routing Station`.

```
Logger0;0.199;1;Class0;0.000;0.199
Logger0;0.559;2;Class0;0.341;0.341
```

Routing Section
For each class of customers, the user should select the algorithm that want to apply in order to determine the outgoing connection (i.e., the routing strategy) followed by a customer that completed its service in the current station. The following algorithms are available:
- Random
- Round Robin (with and without weights)
- Probabilities
- Join the Shortest Queue (JSQ)
- Shortest Response Time

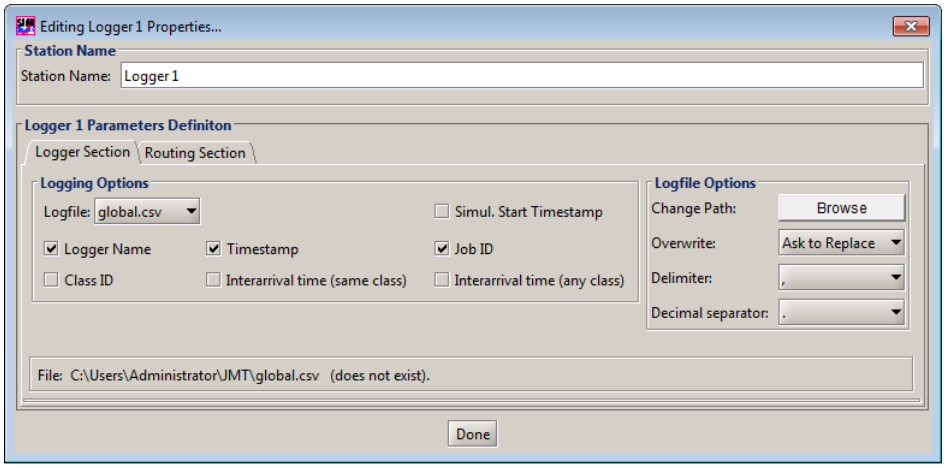


Figure 3.83: Window for the `Logger Station` configuration.

- Least Utilization
- Fastest Service
- Load Dependent Routing
- Power of k choices (with and without memory)
- Disabled.

To know details about these algorithms, please refer to the `Routing Section` of subsection 3.11.1-*Source Station*.

3.11.9 Class-switch Station

A `Class Switch` station allows the definition of models where jobs may change *class*, i.e., *service requests*, during their execution. The switch of the jobs between the classes is defined by the *class-switch probability matrix*, referred to as **C**. Let $R = (1, \ldots, r)$ be the set of the classes of the model, then **C** is a $r \times r$ square matrix. The element $c_{i,j}$, $i, j \leq r$, is the probability that a job entering the `Class Switch` station in class i exit the station in class j.

In Fig.3.84 it is shown an example of a open model with a three-class workload (A,B,C), two `Class Switch` stations, and three `Delay` stations. Jobs are generated by `Source 1` in *class A* and receive service from station `Delay1`. Then, after being served, it changes class from A to B in `Class Switch A->B` station (see the matrix of Fig.3.86) and receive service in `Delay2` station. Then, after being served, it changes again class from B to C in `Class Switch B->C` station and exit the model in *class C*.

Let us remark that when this type of station is included in a model, a ***new*** class of jobs may be *generated* not only from a `Source` station (or in the initial population vector in the case of a closed model) but ***also*** in a `Class Switch` station. This is the case of *classes* B and C of the example. For this type of classes, a `Class Switch` station *must be selected* as `Reference Station` and it is *not allowed* to define their generation process since it depends on the arrival patterns from the other stations (see Fig.3.85). In the model of Fig.3.84, the throughput of the system consists *only* of *Class C* jobs in spite that *only Class A* jobs are arriving at the model generated from `Source1`.

In the last section (*Definition of the context and Implementation issues*) the topic of *class switch* is addressed in more detail by analyzing some problems of context definition and some implementation issues.

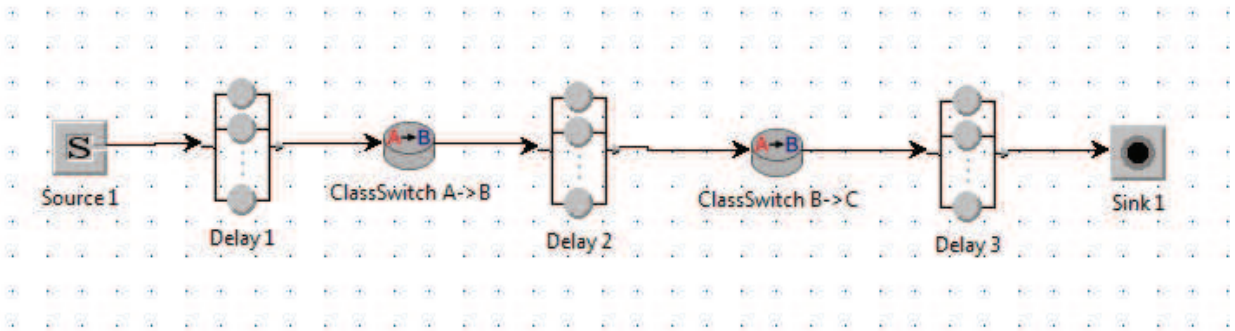


Figure 3.84: Example of an open model with two `Class Switch` stations.

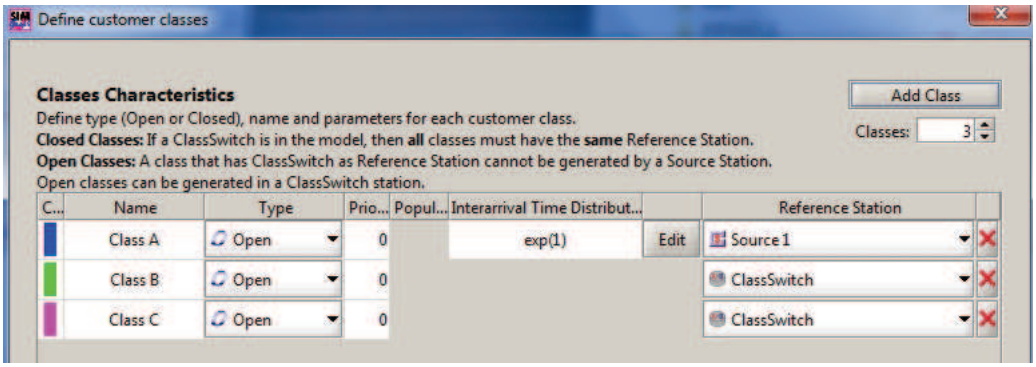


Figure 3.85: Example of the definition of the three-class workload of Fig.3.84, with one class (`Class A`) arriving from `Source 1` station and two classes, `Class B` and `Class C`, generated by `Class Switch` stations.

Set or change the class-switch parameters

Double click on the `Class-switch` icon to open the `Editing Class-Switch Properties` station's configuration dialog (see Fig.3.86). There are two tabs: `Class-Switch Matrix` and `Routing Section`.

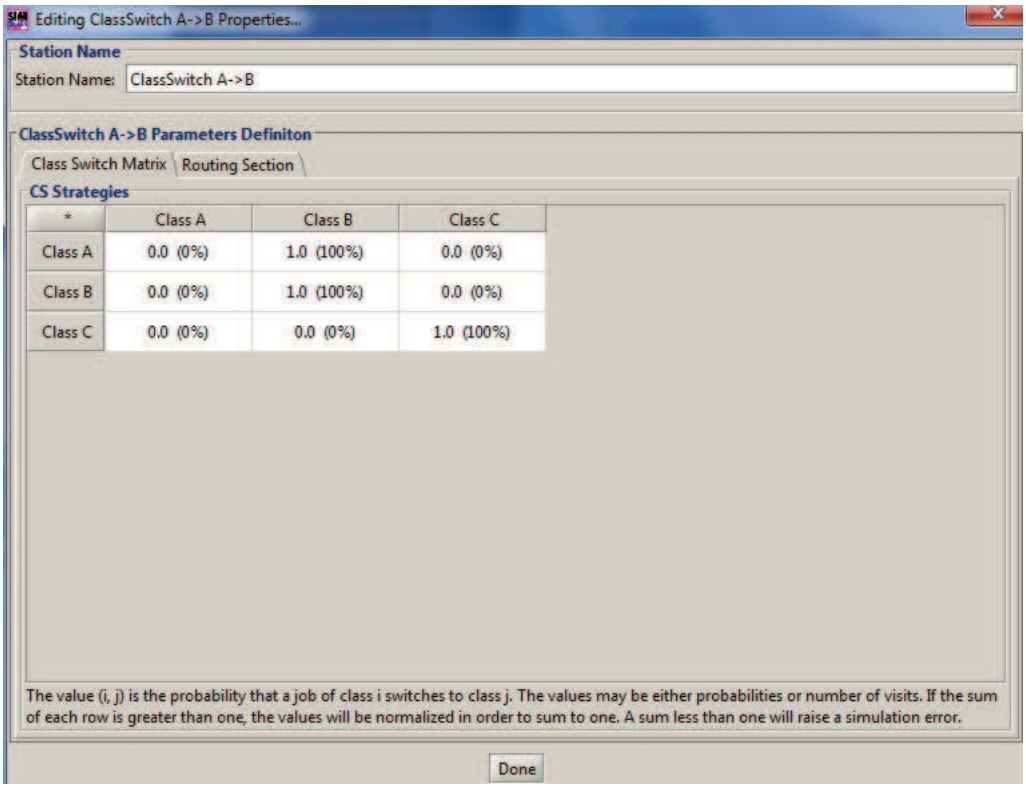


Figure 3.86: Window of the `Class-switch A->B` station for the definition of the transition matrix elements, i.e., of the probabilities of switch between classes, of the model of Fig.3.84.

Class-Switch Matrix

In this section the probabilities of class-switch of the jobs must be described in matrix **C**. Each element of the matrix represents either normalized or absolute values (e.g., number of visits), the sum of each row is constrained to be greater or equal than 1. A *warning* message is given when the sum of the values is greater than 1. An *error* message is prompted when the sum is less than 1, and the simulation does not start (see Fig.3.87).

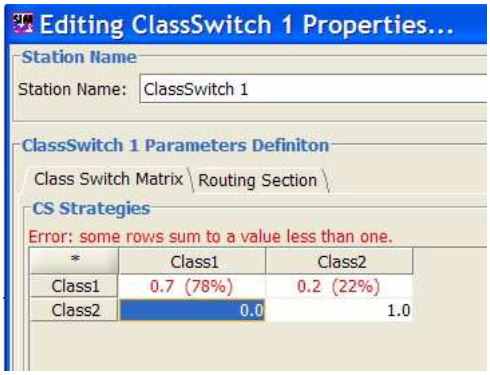


Figure 3.87: Error message prompted by the control of the switching probabilities of `Class1` that sum to a value < 1.

Routing Section

For each class of customers, the user should select the algorithm that want to apply in order to determine the outgoing connection (i.e., the routing strategy) followed by a customer that completed its service in the current station. The following algorithms are available:

- Random

- Round Robin (with and without weights)
- Probabilities
- Join the Shortest Queue (JSQ)
- Shortest Response Time
- Least Utilization
- Fastest Service
- Load Dependent routing
- Power of k choices (with and without memory)
- Disabled.

To know details about these algorithms, please refer to the `Routing Section` of subsection 3.11.1-*Source Station*.

Limitations when using Class-Switch

Several *restrictions* are introduced in the models that comprise one or more `Class Switch` stations.
- `per class System Throughput`: this index is NOT computed since it may be that one or more classes never exit a model due to the combined actions of the `Class Switch` stations and the Routing policies (see the classes A and B of Fig.3.84). Also, it may happens that in a model at the equilibrium the mix of classes in input to the system is not the same as the one in output;
- a *new class* may be generated ALSO in a `Class Switch` station. In this case, a `Class Switch` station must be defined as `Reference Station` and IT IS NOT *allowed* to define an *interarrival* process for its jobs;
- it is NOT POSSIBLE to have a switch of jobs between an *open class* and a *closed class*, and viceversa;
- FOR THE RELEASE 0.9.2: it is not possible to have models with `Class Switch` and `Fork/Join` or `Finite Capacity Region` stations. Sorry!

Definition of the context and Implementation issues of the Class-switch

Models with class switch are very powerful in terms of representativeness, but it must be used very carefully because of the new problems that arise over the traditional models that we are used to.
In what follows we will consider only open models as for them the effects of class switch can be described more intuitively than for the closed ones. However, the validity of the concepts introduced can be easily extended to closed models also. In this case, it is simply required to replace the `Source` station with the `Reference station` and the `arrival process` with the `initial population mix` of the model.
First of all, the *concept of equilibrium* in models with class switch is valid only at the *system level*, and not at the class level. In other words, Little law can be applied only to the level of the *overall system* and not to single class level. Indeed, the population mix of the jobs in the various classes may vary after a class switch station where jobs change class, new classes may be generated or existing classes may disappear. Thus, it may happens that the classes that exit the model are different from the ones that arrive to the model (see e.g., Fig.3.84) violating the basic concept of equilibrium.
Let us remind the difference between the `Service Times` S and the total `Service Demands` D of a job, since they have a direct impact on the difference between `Response Times` (Rp) and `Residence Times` (Rd) of jobs. The `Service time` S_r is the average time that resource r requires to execute a request (i.e., is the mean service time required by *one visit* to that resource). The `Service Demand` D_r of a job at resource r is the *total* service time required to that resource during its complete execution. D_r is given by $V_r S_r$ where V_r is the mean number of visits that a job makes to that resource, also referred to as *visit count*, during its complete execution.
In *single class* models, the `Response Time` Rp_r of a resource r is the mean time required from (*one*) visit to resource r (queue time + execution time). The `Residence Time` Rd_r of a job at resource r is the *global time* spent by the job on that resource during its complete execution. While the metric `Response Time` Rp_r is *local* to a resource (it may be computed considering *only* variables related to the resource considered: queue time and service time), the `Residence Time` Rd_r of a resource is *global*. indeed, to derive its value it is necessary to know the behavior of a job through *all* the resources during its complete execution since the *visit ratio* (also referred to as *visit count*) to all resources must be known.
In *multiclass* models also the `Response Time` Rp_r of a resource r becomes a *global* metric. Indeed, with a workload consisting of C *classes* of jobs, the Rp_r is given by:

$$Rp_r = \sum_{c=1}^{C} Rp_r^c \frac{X_c^{Sys}}{X_{tot}^{Sys}} \tag{3.1}$$

where Rp_r^c is the `Response Time` of the resource r for the jobs of class c, X_c^{Sys} is the throughput of the system of jobs of class c, and X_{tot}^{Sys} is the global `System Throughput`.
The dependence of Rp_r from the per-class throughput makes it unfeasible to apply eq.3.1 for its computation

since with class switch the *per-class* system throughput *cannot be defined precisely.*

IMPLEMENTATION ISSUES
Three COUNTERS are used by JSIM to measure the `System Response Time`, the `System Throughput`, and the `per-class System Response Time`.
The *global* `System Response Time` Rp^{Sys} is obtained by accumulating in a counter the time that all jobs spent in the system during a complete execution and dividing this value by the number of jobs completed (independently from the classes they belong to). The value obtained is *correct* and is independent of the classes of the jobs since the execution time is collected for *all* the jobs completed.
Similarly, the *global* `System Throughput` is obtained by collecting the global number of jobs executed (that exit the model) and dividing its value by the global simulation time. As already pointed out, the `per-class Throughput` cannot be computed in the same way since a job **may change class** during its execution.
To avoid an excessive overload of the memory and of the processing time, 64 bits are associated to the internal data structure of each job. One bit is associated to each class assumed by a job during its execution (thus, 64 is the max number of classes that a job can assume during a complete execution). For each job we keep track *only* of the classes assumed during its complete execution, but NOT of the *sequence* of the classes assumed. This decision has some *positive* aspects related to the size of the memory required to store the traces of the executed jobs (that can be even several millions), and some *negative* aspects due to the limitations of the metrics that can be measured. For *each class* there is a *counter* that keeps track of the number of jobs of that class (i.e., that assumed that class during their execution or that are generated by a Class Switch). So, for the same *single* job it may be that *several* counters will be incremented, as a function of all the classes assumed during its execution. For this motivation, *these counters* are used *only* to compute the `per-class System Response Time` and **NOT** the `throughput`.
To count the jobs in the model, for each class there are *two counters*: *one* in the `Reference Station` (in open models the `Source` is assumed as `Reference Station`, in closed models any station can be a `Reference Station`) and *one* in the `Class Switch` stations. The sum of the jobs of a class that flow out by these two station types represents the *global number of jobs* that *entered* an open model, or that *are* in a closed model. In eq.3.2 we will refer to these jobs as *Generated*.
To compute the `System Response Time` of class A jobs, i.e., the total time spent in the system by a *class A* job during its complete execution (mean value), we need to sum the `Residence Times` of class A jobs for each resource. And, to compute the latter index we need to know the number of times that class A jobs visit each station. Let us remark again that with `Class Switch`, a job may be generated *inside* the model and thus it may *not* visit the *same set* of stations that jobs of the same class have already visited.
To compute the correct number of *visits* that a class A job perform to station r we will compute for each station the following *Visit Ratio*:

$$Visit\ Ratio_A^r = \frac{V_A^r}{Generated\ A} \tag{3.2}$$

where V_A^r is the total number of visits of class A jobs (*all* the class A jobs!) that have gone through station r (collected in a counter) and *Generated A* is the total number of class A that have been generated in the model according to the above definition (by `Source + Class Switch` stations). Then, the `Residence Time` of *class A* jobs at station r will be given by:

$$Residence\,Time_A^r = Response\,Time_A^r * Visit\ Ratio_A^r \tag{3.3}$$

where the `Response Time`$_A^r$ is the mean time required by one visit from a class A job to station r. The sum of all the values given by eq.3.3 for *all* the classes and *all* the stations is the `System Response Time`:

$$System\,Response Time = \sum_{A\in classes} \sum_{r\in stations} Residence Time_A^r * \frac{Generated\ A}{X_0\ or\ All\ Generated\ by\ Source} \tag{3.4}$$

where X_0 should be considered in closed classes and the Generated by Source in open classes. The value obtained with eq.3.4 must coincide with the value measured by JSIM in the corresponding counter summing the `system response times` of all the jobs executed and dividing the result by their number.

3.11.10 Semaphore Station

This station works on **tasks** and thus **must** be located between a `Fork` and a `Join`. It may be selected with the button  of the initial window of JSIMgraph. It blocks the arriving *tasks* of a job generated by the `Fork` until a *threshold* value, assigned as input parameter, is reached (see Fig.3.88). When the threshold is reached, all the blocked tasks of that job are released simultaneously. All the following tasks of the same job will not be

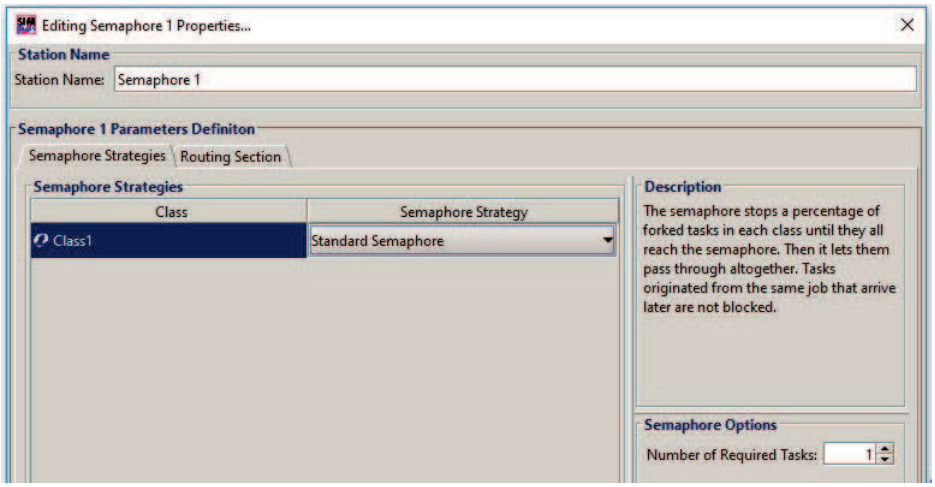


Figure 3.88: Window of the `Semaphore` for the definition of the threshold value (`Number of Required Tasks`).

stopped.
It has no service time, but it may have a response time different from zero because, usually, as a function of the layout of the network some tasks are blocked waiting that the threshold value is reached.
It is important to point out that the control for the threshold is activated only on the tasks that a job has generated at the Fork. The threshold control is performed for each job. The tasks are released by the `Semaphore` according to the order they they reach the threshold. Thus the sequence of release may be different from that followed by the Fork.
The default value of the threshold is 1, and each *class of jobs* has its threshold value. The *max value* of the threshold considered by a Semaphore is the sum of all the tasks generated by the Fork for a job (considering all the output paths).

3.11.11 Scaler Station

This station may be selected with the button  of the initial window of JSIMgraph.
The `Scaler` can be seen as a station that *aggregate* a Join connecting to a Fork. Indeed, a `Scaler` station is composed of a *Join* (input) section, a *Tunnel* (service) section and a *Fork* (output) section. Tasks forked from the same job are joined at the input section of a `Scaler`. After that, the job is propagated immediately to the output section, where it is forked into another group of tasks. Fig.3.89 shows a typical utilization of a `Scaler` in between a `Fork` and a `Join`.

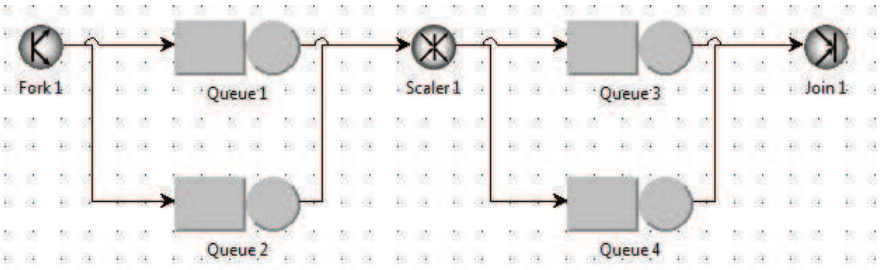


Figure 3.89: Example of a typical utilization of a `Scaler` between a `Fork` and a `Join`.

The *difference* between a Scaler and a Fork is that the input section of the `Scaler` is a `Join section` while that of the `Fork` is a `Queue section`. The *difference* between a `Scaler` and a `Join` is that the output section of the `Scaler` is a `Fork section` while that of the `Join` is a `Routing section`. The sections of the `Scaler` have *exactly the same parameters* as those of the `Fork` and `Join` (see Fig.3.90).

3.11.12 Place and Transition Stations (Petri Nets)

The `Place` ○ and `Transition` ▯ stations have been designed and implemented for simulation of *Petri nets* [Pet62]. With these two stations, JSIM well supports almost all the canonical types of Petri nets including GSPNs (Generalised Stochastic Petri Nets), CPNs (Coloured Petri Nets) and QPNs (Queueing Petri Nets)

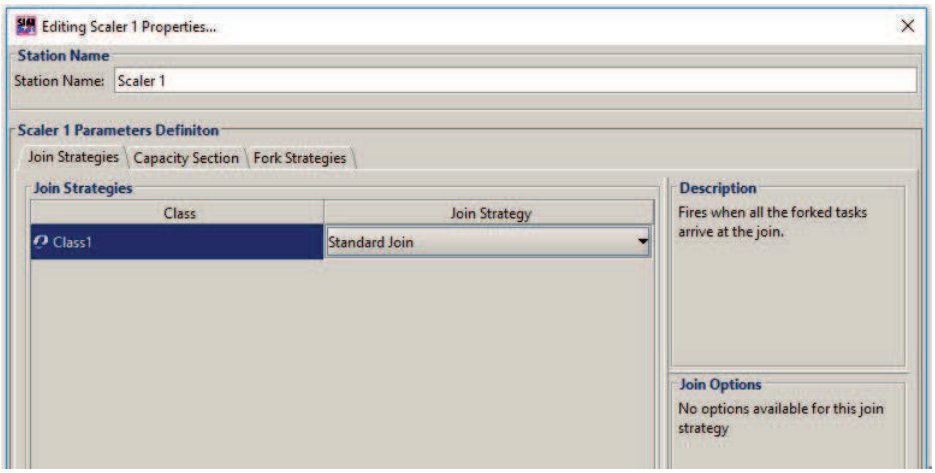


Figure 3.90: Window of the `Scaler` for the definition of the parameters of the aggregated `Join` and `Fork`.

[BK02]. High compatibility has been achieved between the Petri net (PN) and queueing network (QN) stations. Thus, *hybrid models* comprising the PN and QN components can also be simulated with JSIM. In hybrid models, the PN components are especially useful for describing the logical behaviour while the QN components are particularly suitable for characterising the queueing behaviour.
The PN stations are compliant with the three-section structure adopted by the QN stations, i.e., input, service and output sections. The `Place` station consists of `Storage`, `Tunnel` and `Linkage` sections, and the `Transition` station is composed of `Enabling`, `Timing` and `Firing` sections. Users may specify the parameters of `Storage`, `Enabling`, `Timing` and `Firing` sections. Fig.3.91 gives an example of a simple PN model with two closed classes. Let us consider this example to see what parameters are provided by these sections.

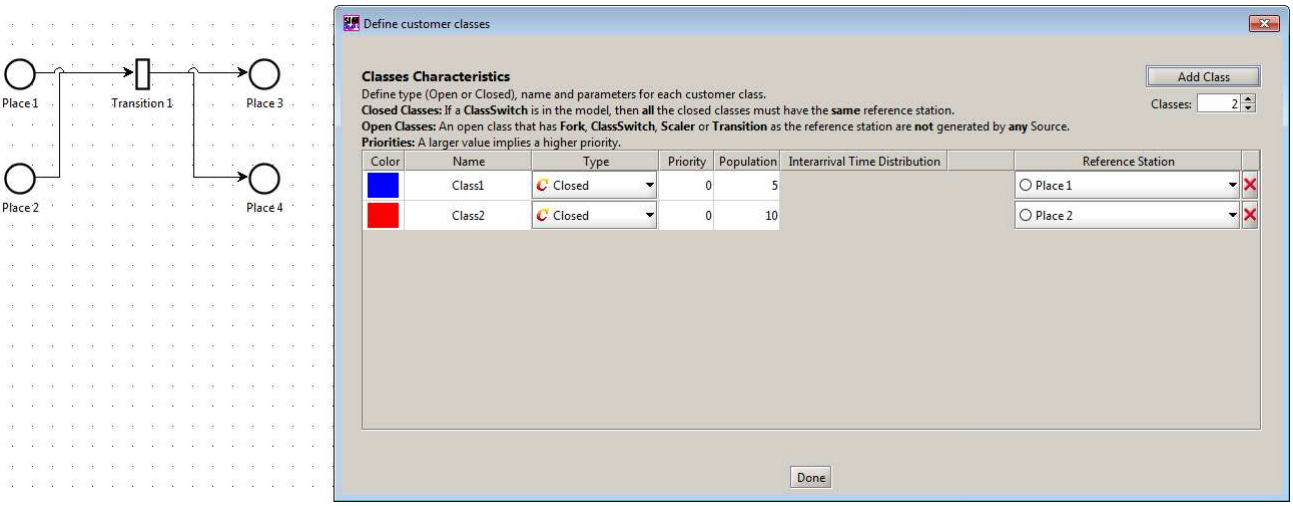


Figure 3.91: Example of a simple PN model with two closed classes.

The `Storage` section panel shown in Fig.3.92 allows users to specify the *storage capacities* and the *storage strategies* of Place 1. The storage capacities can be either `Infinite` or `Finite` for all the classes as well as for each class. The storage strategies include the *queue policies* and the *drop rules*, and can be specified separately for each class. Three queue policies, `FCFS`, `LCFS` and `Random`, are applicable to the `Storage` section. The available drop rules are `Drop`, `BAS Blocking` and `Waiting Queue`, which can be selected in case of `Finite` storage capacities.
With the `Enabling` section panel shown in Fig.3.93, users can specify the *enabling condition* and the *inhibiting condition* of Transition 1 in two tables. Each row of the tables represents an input station of Transition 1, and each column corresponds to a defined job class. Therefore, an entry of the enabling table indicates the number of jobs of a class required at an input station for enabling Transition 1. Similarly, an entry of the inhibiting table indicates the number of jobs of a class required at an input station for inhibiting Transition 1. Please note that for the inhibiting table an input value of 0 means infinity.
As shown in Fig.3.94, the `Timing` section panel are used to specify the *number of servers* and the *timing strategy* of Transition 1. The number of servers can be either `Infinite` or `Finite`. Two timing strategies are

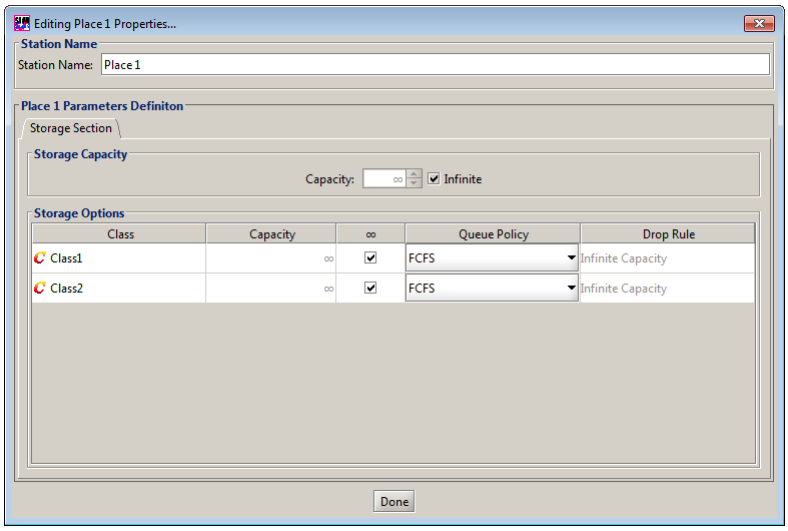


Figure 3.92: `Storage` section panel of Place 1.

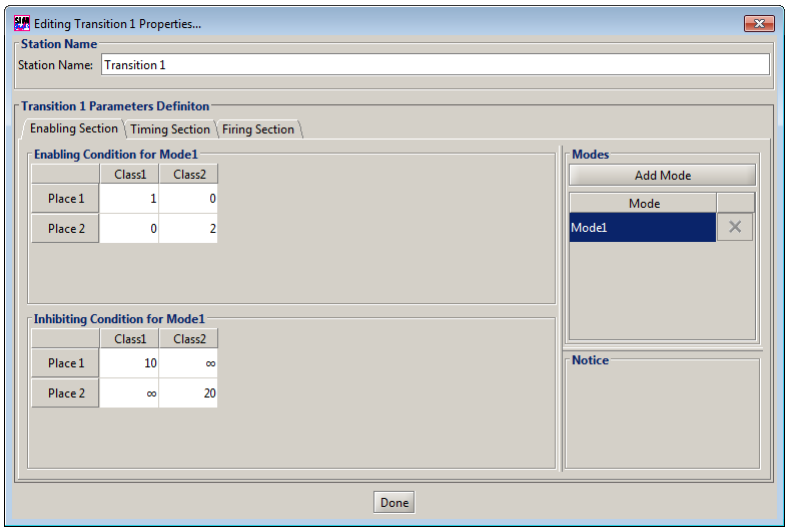


Figure 3.93: `Enabling` section panel of Transition 1.

available including `Timed` and `Immediate` strategies. With the `Timed` strategy, users may specify the *firing time distribution* for which all the implemented distributions are applicable. The *firing priority* and the *firing weight* are enabled when the `Immediate` strategy is selected.
Fig.3.95 shows the `Firing` section panel which allows users to specify the *firing outcome* of Transition 1 in a table. Each row of the table represents an output station of Transition 1, and each column corresponds to a defined job class. Therefore, an entry of the table indicates the number of jobs of a class released to an output station on firing Transition 1.

The introduction of the Petri net theory is beyond the scope of this manual. Please refer to [BK02] or other books for more details about the basic concepts such as *place*, *timed/immediate transition*, *token*, *input/output/inhibitor arc*, *enabling rule*, *firing rule*, *marking*, etc. The following paragraphs are dedicated to clarifying several advanced issues regarding the design and implementation of the `Place` and `Transition` stations. Some examples of PN and hybrid models are also provided in the JMT working folder to help with a better understanding of how to apply these two stations.

Multiple Enablings of a Transition
It is possible that a transition is enabled more than once, as multiple enabling sets of tokens may be present in the input places. The number of times that a transition is enabled, is referred to as the *enabling degree*. Formally, transition t has enabling degree k at marking M if and only if

- $\forall p \in {}^\bullet t,\ M(p) \geq kI(p,t)$

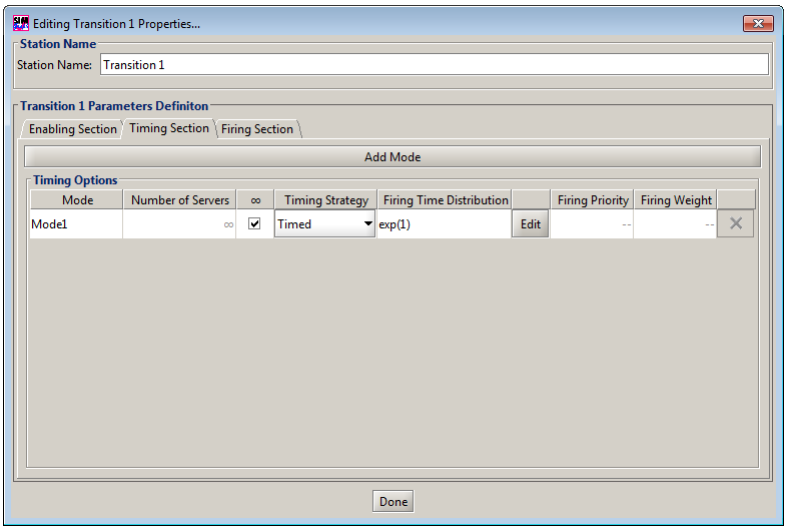


Figure 3.94: `Timing` section panel of Transition 1.

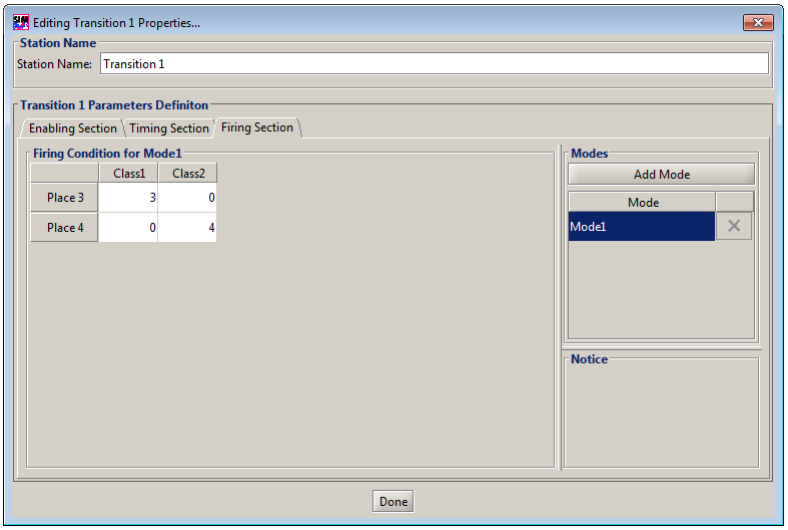


Figure 3.95: `Firing` section panel of Transition 1.

- $\exists p \in {}^\bullet t,\ M(p) < (k+1)I(p,t)$

where p denotes a place, ${}^\bullet t$ is the set of input places of transition t, $M(p)$ is the number of tokens in place p at marking M, and $I(p,t)$ is the number of tokens required in place p for enabling transition t [MBC+95]. When a `Transition` station is enabled, its enabling degree is taken into account.
Different meanings are possible when multiple enabling sets of tokens form in the input places of a transition. Thus, special attention must be paid to the *timing semantics* of a transition with enabling degree greater than 1. Similar to a queue, a transition can have three different timing semantics [MBC+95]:

- **Single-server Semantics:** There is a single server in the transition. Multiple enabling sets of tokens are processed serially.
- **Multiple-server Semantics:** There are multiple servers in the transition. Multiple enabling sets of tokens can be processed in parallel, but the degree of parallelism is limited by the number of servers.
- **Infinite-server Semantics:** There are infinite number of servers in the transition. Multiple enabling sets of tokens are processed in parallel regardless of the degree of parallelism.

All the three timing semantics are supported by the `Transition` station. Notably, the single-server and multiple-server semantics are combined together into the finite-server semantics.

Simultaneous Firings of Transitions
JSIM is equipped with a discrete-event simulation engine called JSIM*engine*. The core component of JSIM*engine*

is an *event calendar*, which works as a message dispatcher between simulation entities [BCS07]. Messages scheduled for the same simulation time are dispatched in the order that they are created on the event calendar. This simple mechanism can result in disruption to the simulation when events completed by sequences of messages are dependent and simultaneous, because messages belonging to different sequences are interleaved arbitrarily so that dependencies between the events are likely to be broken. The firing of a `Transition` station is an event that must be completed by a sequence of messages. If multiple `Transition` stations that share some `Place` stations as their inputs happen to fire at the same simulation time, communication between the `Place` and `Transition` stations may probably fail to coordinate the simulation.
In order to solve the problem, a reordering mechanism has been implemented in JSIM for dispatching the messages. Particularly, this mechanism gives two properties to the firing event:

- **Atomicity:** The sequence of messages completing each firing event is dispatched as if it is atomic.
- **Certainty:** Any firing event is processed only when no other types of events are left at the same simulation time.

Besides, the reordering mechanism also determines the processing order of simultaneous firing events according to their priorities and weights.

Multiple Colours of Tokens and Multiple Modes of a Transition
Multiple classes of jobs can be defined for PN models. The classes of jobs in QNs conceptually correspond to the *colours* of tokens in CPNs. However, critical distinctions between these two concepts must be pointed out. Jobs waiting in a queue may be sorted by the priorities of their classes, while tokens stored in a place are usually treated equally regardless of their colours. QNs allow servers in a queue to process jobs separately or simultaneously, and the service time distributions are associated with the classes of the jobs. By contrast, CPNs allow a transition to work in different *modes*, and each mode determines an enabling condition, an inhibiting condition, a timing strategy and a firing outcome.
To meet these distinctions, only the `Non-preemptive` discipline is applicable to the input section of the `Place` station, i.e, the `Storage` section. Further, the `Transition` station is designed as a multi-mode transition that can process multiple colours of tokens. As a result, any GSPN model is compressible into a CPN model with a single pair of `Place` and `Transition` stations by applying the following correspondence:

- Tokens at different `Place` stations in the GSPN model correspond to tokens of different colours at a `Place` station in the CPN model.
- Different `Transition` stations in the GSPN model correspond to different modes of a `Transition` station in the CPN model.

Compatibility between PN and QN Stations
QPNs are a hybrid formalism that reconciles PNs with QNs by introducing the *queueing place* [Bau93]. As illustrated in Fig.3.96, a queueing place is composed of two parts: a queue and a depository. Any token released to the queueing place by the input transitions, enters the queue according to the scheduling discipline. Tokens in the queue are invisible to the output transitions. After completion of its execution, a token immediately moves to the depository, where it becomes available to the output transitions.

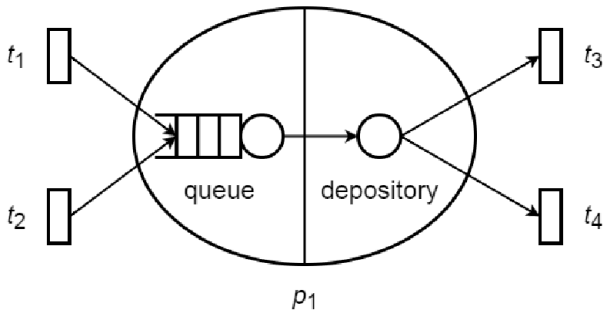


Figure 3.96: Structure of a queueing place.

With compatibility between the PN and QN stations, a queueing place is equivalent to a `Queue` station connecting to a `Place` station. Other hybrid structures can also be easily obtained by combining different PN and QN stations together. However, the following constraints are imposed on connection of the PN stations:

- A `Place` station can only be connected to `Transition` stations.
- A `Transition` station can only be connected from `Place` stations.

These constraints imply that any station is connectable to a `Place` station or from a `Transition` station. Full compatibility between the PN and QN stations has not been achieved in terms of certain strategies and performance indices. For example, since `BAS blocking` can easily introduce deadlocks into PN models, the PN stations are unblockable in the current implementation. The *utilisation* of the PN stations is also not available at present, as an agreement on its definition has not been reached yet.

Import/Export of PNML models

PNML (Petri Net Markup Language, http://www.pnml.org/index.php) is a proposal of an XML-based interchange format for Petri nets. ISO/IEC 15909 (Systems and software engineering – High-level Petri nets) is dedicated to standardising the PNML format. This standard consists of two parts: ISO/IEC 15909-1:2004+A1:2010 Part 1 formulates the concepts, definitions and graphical notation of Petri nets; ISO/IEC 15909-2:2011 Part 2 elaborates the specification of the transfer format on the basis of Part 1.

Four types of PNML models are defined by the standard, including *core models*, *Place/Transition nets*, *symmetric nets* and *high-level Petri nets*. In the current implementation, JSIM only supports import/export of core models and Place/Transition nets. The former are used to convey the topologies of PN models, while the later have been extended to represent GSPN models. Users can import/export a PNML model by simply opening/saving the model from/into a file suffixed with '.pnml'. The type of PNML model to be exported depends on the number of closed classes in the model, i.e., 0 for core models and 1 for Place/Transition nets. However, exporting a PNML model with open classes or PN stations is not allowed.

As illustrated in Listings 3.1–3.5, tool specific parameters are applicable to the `Net`, `Place`, `Transition` and `Arc` elements of a PNML model. These parameters enable the model to cover the extra PN features provided by JSIM. An exported PNML model will contain a complete set of tool specific parameters for every element. As for an imported PNML model, this is not necessary and a default value will be set for each absent parameter.

Listing 3.1: Tool specific parameters for the `Net` element.

```
<toolspecific tool="Java Modelling Tools - JSIM" version="1.0.1">
    <tokens>
        <!-- Data Type: String,
             Valid Values: all the valid strings,
             Default Value: "Token" -->
        <className>Token</className>
        <!-- Data Type: String,
             Valid Values: all the node names,
             Default Value: N/A -->
        <referenceNode>Place 1</referenceNode>
        <graphics>
            <!-- Data Type: Long(hex),
                 Valid Values: [#00000000,#FFFFFFFF],
                 Default Value: #FF0000FF -->
            <color>#FF0000FF</color>
        </graphics>
    </tokens>
</toolspecific>
```

Listing 3.2: Tool specific parameters for the `Place` element.

```
<toolspecific tool="Java Modelling Tools - JSIM" version="1.0.1">
    <!-- Data Type: Integer,
         Valid Values: {-1}U[1,Integer.MAX_VALUE] (-1=infinity),
         Default Value: -1 -->
    <capacity>-1</capacity>
    <graphics>
        <!-- Data Type: Boolean,
             Valid Values: {false,true},
             Default Value: false -->
        <rotate>false</rotate>
    </graphics>
</toolspecific>
```

Listing 3.3: Tool specific parameters for the `Timed Transition` element.

```
<toolspecific tool="Java Modelling Tools - JSIM" version="1.0.1">
    <!-- Data Type: Integer,
         Valid Values: {-1}U[1,Integer.MAX_VALUE] (-1=infinity),
```

```
        Default Value: -1 -->
    <numberOfServers>-1</numberOfServers>
    <!-- Data Type: String,
         Valid Values: {"Timed","Immediate"},
         Default Value: "Timed" -->
    <timingStrategy>Timed</timingStrategy>
    <!-- Data Type: Distribution,
         Valid Values: all the available distributions,
         Default Value: Exponential(1.0) -->
    <firingTimeDistribution>
        <distribution>
            <type>Exponential</type>
            <parameter>
                <name>lambda</name>
                <value>1.0</value>
            </parameter>
        </distribution>
    </firingTimeDistribution>
    <graphics>
        <!-- Data Type: Boolean,
             Valid Values: {false,true},
             Default Value: false -->
        <rotate>false</rotate>
    </graphics>
</toolspecific>
```

Listing 3.4: Tool specific parameters for the `Immediate Transition` element.

```
<toolspecific tool="Java Modelling Tools - JSIM" version="1.0.1">
    <!-- Data Type: Integer,
         Valid Values: {-1}U[1,Integer.MAX_VALUE] (-1=infinity),
         Default Value: -1 -->
    <numberOfServers>-1</numberOfServers>
    <!-- Data Type: String,
         Valid Values: {"Timed","Immediate"},
         Default Value: "Timed" -->
    <timingStrategy>Immediate</timingStrategy>
    <!-- Data Type: Integer,
         Valid Values: [0,Integer.MAX_VALUE],
         Default Value: 0 -->
    <firingPriority>0</firingPriority>
    <!-- Data Type: Double,
         Valid Values: (0.0,Double.MAX_VALUE],
         Default Value: 1.0 -->
    <firingWeight>1.0</firingWeight>
    <graphics>
        <!-- Data Type: Boolean,
             Valid Values: {false,true},
             Default Value: false -->
        <rotate>false</rotate>
    </graphics>
</toolspecific>
```

Listing 3.5: Tool specific parameters for the `Arc` element.

```
<toolspecific tool="Java Modelling Tools - JSIM" version="1.0.1">
    <!-- Data Type: String,
         Valid Values: {"Normal","Inhibitor"},
         Default Value: "Normal" -->
    <type>Normal</type>
</toolspecific>
```

3.12 Modification of the Parameters

3.12.1 Modifying simulation parameters

To modify the simulation or initial state parameters, the user has to select `Simulation Parameters` (icon ) from the `Define` menu and change the values. From `Simulation Parameters` the user can:

1. change the *initial state*, i.e., the location of the customers at the beginning of the simulation
2. change or confirm the *seed* of the random number algorithm in order to have different run of the simulation
3. change the maximum number of *samples* at steady-state for the performance indexes
4. change the maximum number of *events* allowed to occur in the simulation since initialization (irrespectively of whether they contribute to a performance index or not)

5. change the time interval used in the graphical representation of the indices
6. change or confirm the maximum *duration* of the simulation
7. change or confirm the number of *parallel threads* used in the What-If analysis.

For more details about these parameters see the Section `Simulation Parameters` of subsection 3.12.3.

3.12.2 Modifying station parameters

To modify the parameters of a station select the icon and double click on the station to modify. The properties panel of a station will be opened and the modifications could be applied.
To see a complete list of properties of each station see section 3.11 *Defining Network Topology.*

3.12.3 Modifying the default values of parameters

The simulator default settings for *Class, Station, Simulation,* and *Finite Capacity Region* parameters can be changed from the `Define` menu in the menu bar. At first the user need to select `Default parameters` from the `Define` menu. Alternatively, the user can select the (`Define default values` of model parameters)

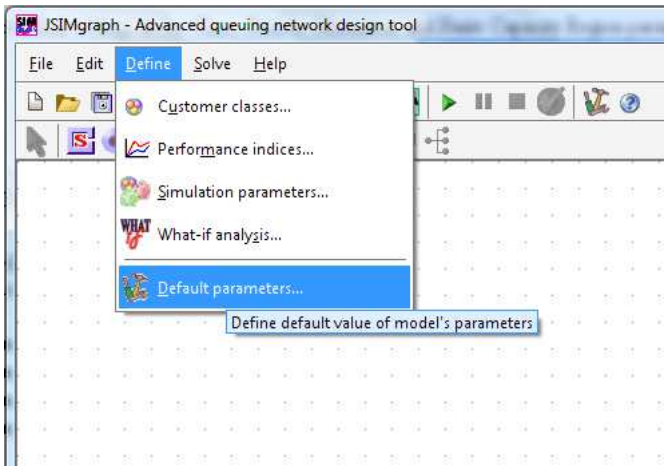


Figure 3.97: Selection of the `Default parameters` from the `Define` menu.

button. In JSIMgraph all the parameters have default values. Such values can be changed to suit the user's most common modelling activities. The default values of *Class, Station, Simulation, Finite Capacity Region (FCR)* and their possible modifications are described in the following sections.

Class Parameters
These settings define the default values of class parameters assigned to a new class (see Figure 3.98).

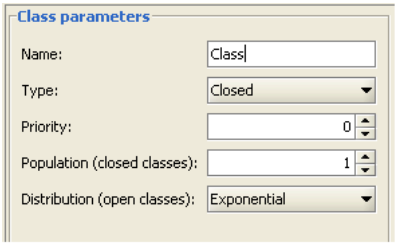


Figure 3.98: The `Class parameters` modification window of default values.

- *Name:* The default name for each newly created class of customers is "*Class*", followed by a progressive integer, starting from 1, indicating how many classes you have created so far. It can be set to any alphanumerical string, which automatically will be followed by the same numbering scheme.
- *Type:* The default class type is "*Closed*". If open classes are used more often, you may want to change to Open as default, so as to minimize the type changes in the class definition section.
- *Priority:* The default class priority is 0. Higher numbers correspond to higher priority.

- *Population:* This parameter applies *only* to closed classes; the default value is 1, i.e., all closed classes are created with 1 customer each. Change the value if you want larger populations by default in each closed class.
- *Distribution:* This parameter applies *only* to open classes; the default value is Exponential since it is the most popular distribution that characterizes customer interarrival times at a system. Change it to any of the distributions available if most of your customer interarrival times follow a non-exponential pattern.

Station Parameters
These settings are general parameters for any type of station (see Figure 3.99).

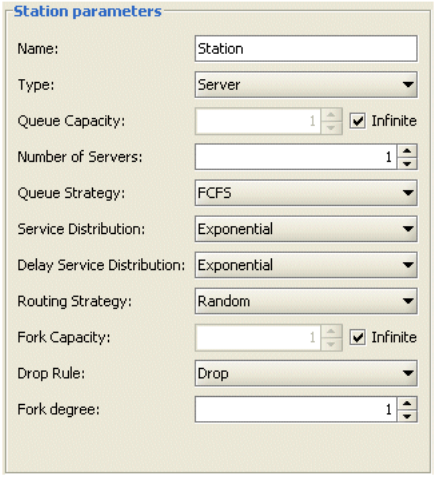


Figure 3.99: `Station parameters` modification window of default values.

- *Name:* the default name of each newly created station is "*Station*", followed by a progressive integer, starting from 1, indicating how many classes you have created so far. It can be set to any alphanumerical string, which will be followed by the same numbering scheme.
- *Type:* the default station type is "*Queueing*" (sometimes referred to as Server), since this is the most popular type of station in performance models. Alternative types are Fork, Join, Routing, Delay, Logger.
- *Queue Capacity:* this parameter applies only to Queueing and Fork station types; the default value is *infinite*, i.e., an infinite number of customers can queue at such stations. It can be set to finite, with a finite value specified, by checking out the `Infinite` checkbox.
- *Number of Servers:* this parameter applies only to Queueing stations; the default value is 1, as most devices are single server. It can be changed to any finite value.
- *Queue Strategy:* this parameter applies only to Queue and Fork station types; the default value is FCFS and can be changed to LCFS. In both cases, a priority version of the discipline is also available.
- *Service Distribution:* this parameter applies only to Queueing Stations; the default value is Exponential. It can be changed to any of the available distributions if most of the devices modelled are non-exponential.
- *Delay Service Distribution:* this parameter applies only to Delay stations; the default value is Exponential but i can be changed to any of the distributions available.
- *Routing Strategy:* this parameter applies only to Delay, Queueing, Join and Routing stations; the default value is Random and it can be changed to any of the available routing strategies, namely Random, Round Robin (with and without weights), Probabilities, Join the Shortest Queue, Shortest Response Time, Least Utilization, Fastest Service, Load Dependent Routing, Power of k choices (with and without memory), and Disabled.
- *Drop Rule:* this parameter applies only to Queueing and Fork station types. When the capacity is infinite, the the Drop Rule is disabled by default. When a finite capacity is defined, the following policies are possible. Three other policies are possible:
  - **Drop** is the default option, i.e., all customers exceeding the station's buffer finite capacity are discarded upon arrival.
  - **Waiting Queue** means that the specified class actually has an infinite waiting buffer. This option allows to specify hybrid queues where the finite capacity applies only to a subset of the classes that excludes those with Waiting Queue enabled.

  - **Blocking After Service (BAS)** blocks the customers in the station before the one considered if its maximum capacity is reached. BAS implements the FBFU policy, First Blocked First Unblocked, which means that each target queue keeps a separate list of the BAS blocked queue and when there is empty space unblocks the first in the list.
  - **Retrial** implies that a customer that finds the station full temporarily leaves, going in a virtual place called the **Retrial Orbit**, from which it returns to retry to join the queue after some time. Choosing the Retrial option opens a dialog window to specify the time that a job will spend in the Retrial Orbit, before retrying to join the queue. Dedicated performance measures are available in JMT to study the jobs in the Retrial Orbit, as these do **not** count towards the ordinary metrics such as the *Number of Customers* in the queue. These include: *Retrial Rate* (the mean number of jobs per unit time trying to join the queue from the Retrial Orbit), *Retrial Orbit Size* (the mean number of jobs in the Retrial Orbit), and Retrial Orbit *Residence Time* (the mean time jobs spend in the Retrial Orbit).
- *Fork Capacity:* this parameter applies only to Fork stations; the default value is infinite, i.e., no limit on the number of customers entering a Fork station exists. It can be changed to finite, by checking the blocking checkbox and a finite number must then be specified.
- *Forking Degree:* this parameter applies only to Fork stations; the default value is 1, i.e., one task is generated for each outgoing link of the Fork station. It can be changed to any finite value, thus increasing the degree of parallelism a job has (and the number of tasks it is split into).

Simulation Parameters

These parameters define the simulation run from a statistical point of view. More details on *Confidence Interval* and *Max Relative Error* are reported in subsection 3.7.1 and subsection 3.7.2.

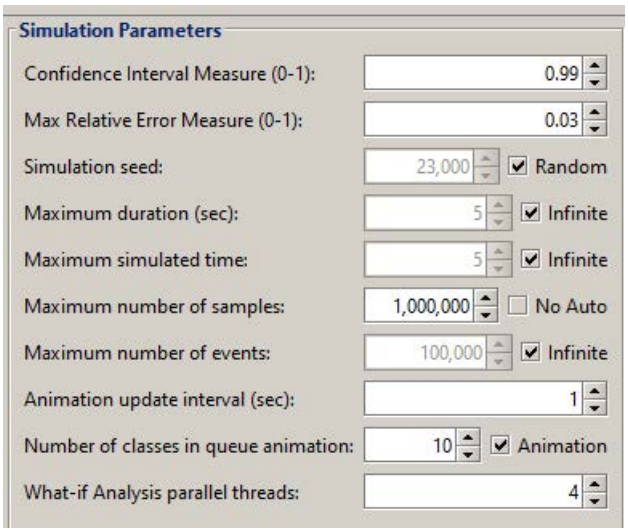


Figure 3.100: Parameters for the control of the simulation.

- `Confidence Interval Measure:` This parameter is set by default at 99%, a commonly used value as it provides a good trade-off between results accuracy and simulation time. Larger values will lead to more accurate results at the expenses of an increase in the time the simulation takes to complete, smaller values will achieve the opposite effect.
- `Max Relative Error Measure:` This parameter is set by default at 3%; smaller values will increase results accuracy while bigger values allow for larger errors in the samples.
- `Simulation Seed:` this parameter is used to initialize the pseudo-random number generator; the generated sequence of pseudo-random numbers is correlated with the seed used. A *Random* value is selected by default (typically related to the clock time), or a user may define its own value.
- `Maximum Duration:` this parameter is set to infinite by default, so that even lengthy simulations can complete and satisfy even narrow confidence intervals; finite values (in seconds) may force termination before the end of the simulation.
- `Maximum Simulated time:` this parameter is set to `Infinite` by default, this is the time simulated during the simulation run. As a function of the objective of the study, a user may want to stop the

simulation at a given time. In this case, the user may uncheck the `Infinite` check box and type the desired value of the simulated time.

- `Maximum Number of Samples:` this parameter is by default handled by the simulation engine automatically. It should be tuned depending on the objectives of the simulation study performed and ensure efficient simulation time. With a fixed number of samples (e.g., 10,000,000) and enabling the "no auto" flag, it may be possible to force the simulation to run long enough to achieve the requested confidence interval in case there automatic stop criteria stops the simulation too early.
- `Maximum Number of Events:` this parameter is disabled by default. With a fixed number of events it is possible to stop the simulation whenever enough events (e.g., arrivals and service) have occurred in the simulation engine. This differs from the number of samples in that it is independent of the performance indexes. For example, if we ask 1,000,000 samples for a throughput for a class of customers that only rarely visits a queue, the simulation may take a long time to finish and we may not know this in advance of running it, which is a problem to plan for the total execution time of many experiments (for example using the what-if analysis). Instead, by fixing the maximum number of events we can ensure that the simulation will approximately take a similar time at every run.
- `Animation Update Interval:` this parameter indicates the time interval between consecutive graph updates on the screen; the default value is 1 sec, as it provides for a smooth progression of the graphs on the screen. Larger values will make the evolution of the simulation appear jerky.
- `Number of classes in queue animation:` this parameter controls the number of classes for which graphs will be plotted for the selected performance indices. By setting the default value to 10, all classes are represented in most models. Graph animation is enabled by default but can be disabled.
- `What-if Analysis parallel threads:` this parameter sets the number of threads that will be used to independently simulate JSIM models during a what-if analysis. The default value is set automatically by JMT to be the number of logical cores of the machine (in the figure: 4), thus if hyper-threading is enabled this value will be larger than if hyper-threading is disabled. Once the default value is manually modified, the same setting will be used again when JSIM is re-started. A user may decide to set this value to a lower number of threads to make sure that the machine is responsive and can smoothly execute other applications while JMT runs a what-if.

Finite Capacity Region (FCR) Parameters

The default values for newly created `Finite Capacity Regions` follows.

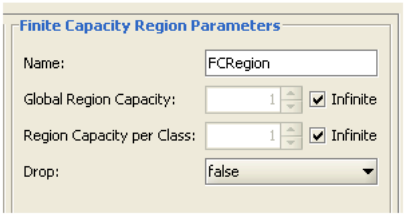


Figure 3.101: Parameters of a `Finite Capacity Region`.

- `Name:` the default name of each newly created Finite Capacity Region is `FCRegion`, followed by a progressive integer, starting from 1, indicating how many classes you have created so far. It can be reset to any alphanumerical string, which will be followed by the same numbering scheme.
- `Global Region Capacity:` the default value of this parameter is infinite, i.e., there is no upper limit to the total number of customers allowed in the region; the value can be changed to finite, and a number must be specified.
- `Region Capacity per Class:` the default value of this parameter is infinite, i.e., there is no upper limit to the per class total number of customers allowed in the region; the value can be changed to finite and a number must be specified.
- `Drop:` the default value of this parameter is `False`, i.e., no customer is ever discarded when arriving at the region; it can be changed to `True`, in which case customers in excess of the region capacity are discarded.

Reset to Default Parameters

If you have modified the default parameter and you want to restore the original values press the `Reset` button at the bottom of the panel.

3.13 Error and Warning Messages

JSIM*graph* analyzes the network before starting the simulation. If errors or warnings are found, a window reporting the description of the errors, like the one of Figure 3.102, is shown.
Click on an error or warning message to open the panel where you can fix it (modifying the wrong parameters or changing the network).
In the following table the *diagnostic messages* of the most common errors and the suggested actions to fix them are shown.

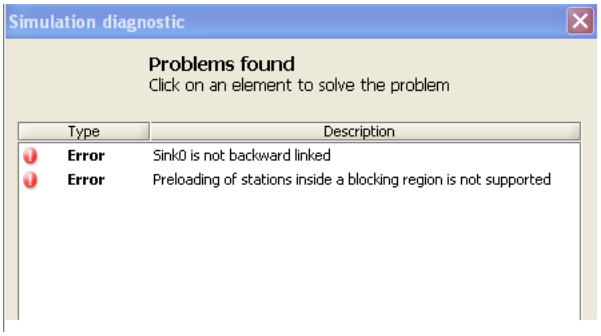


Figure 3.102: Window with some diagnostic errors found by the simulator.

Station X is not forward linked	Error	This error occurs when the station (source or join) has not outgoing, or forward, links to any other station in the model. Only Sink stations do not have forward links.
Station X is not backward linked	Error	Each station must have at least an incoming, or backward, link from other stations, for open class jobs to be able to leave the system. Only the Source station does not have a backward link.
No reference station defined for Class X	Error	For each class you must define a Reference Station that is used to estimate system throughput and response time. Click on this error to open and set the missing reference station.
No performance indices defined	Error	This error occurs when you try to start the simulation but no performance index is defined.
Fork found but no join	Warning	Fork and Join stations should be inserted together in the model. If you have inserted only one of the two stations, when the simulation is run JSIM will show a diagnostic message. Having a Fork without a Join is possible, although it may lead the system to saturate quickly.
Join without fork	Error	Having a Join without a Fork is a mistake, hence the Error message. The missing station must be added with the corresponding links.
Source X without open classes associated	Error	Insert in the network a Source only if you have an open class. If you have only a closed class, the source cannot be defined. But if you have open and closed classes you have to define a Source and a Sink in the network. This error appears when you have built a model with a source but without open customer classes.
No classes defined	Error	The definition of at least one class of customers is required in order to solve a model. Customer classes are a mandatory parameter of any model.
A performance index is defined more than once	Error	In the definition of the model output, the same index has been added more than once for the same station and class. Multiple occurrences must be removed.
Closed class Class X routed to a station linked only to sink	Error	Customers of closed classes keep circulating in the system. They may not visit a Sink station or a station that is connected on the outgoing link only to the Sink, since this would cause them to leave the system. The station connections must be changed.
Undefined station in performance index	Error	When a new performance index is defined, a connected station should also be selected. Click on Performance Index from Define menu and select the correct station.
No station defined	Error	You have to insert at least one station in the model.
Open class found but no source defined	Error	When an open customer class is defined, a Source and a Sink must be added to the network (to generate and collect the jobs). Add a source to the network.
Closed class ClassX at station StationY is routed to a sink with p=1	Error	In the routing section of station StationY, the routing probability of a closed class job to a sink is one. This is an error because jobs cannot leave a closed model. Add another link to connect the station with other components of the network and set the new routing probabilities.
Open classes were found but no sink have been defined	Error	When an open customer class is defined, a Source and a Sink must be added to the network (to generate and collect the jobs). Add a sink to the network.
Sink without open classes	Error	A sink is only used to collect customers of open classes. This means that a sink without any closed class is useless.
More than one sink defined, measures may not be accurate	Warning	The evaluation of System Throughput performance index may be inaccurate with multiple sinks. We recommend to use one sink only.
What-if analysis model modified	Warning	One What-if analysis has been defined but after the model has been modified adding a station. The analysis may not work properly. Redefine the What-if Analysis
Finite Capacity Region RegionX is empty	Error	You have defined a Finite Capacity Region that is empty. A Finite Capacity Region must contain least one station.
Preloading of stations inside a Finite Capacity Region is not supported	Error	This error appears because you have defined some initial states for the stations in the Finite Capacity Region. All Stations in an FCR must have initial state equal to 0 otherwise this error will appear. Preloading is not supported in FCR.
Response Time per Sink and Throughput per sink should not be used if there is no Sink defined in the model	Error	This error appears because you have selected Response Time per Sink or Throughput per Sink performance Index but there no sink is defined in the model.
Response Time per Sink and Throughput per sink should not be used for closed class	Error	This error appears because you have selected Response Time per Sink or Throughput per Sink performance Index with respect to closed class. In closed class models jobs cannot leave the model and hence no jobs can be routed to sink.
In StationX, some rows sum to a vale less than one	Error	This error appears because the sum of the probabilities of switch of a class is less than one in a *class-switch* station.

When you try to Export model defined with JSIM to JMVA some additional errors can appear due to the assumptions that must hold in order to apply the MVA algorithm.

XXX routing strategy in ClassX for Source is not allowed. This was considered as RandomRouting	Warning	A few constraints apply as for the admissible routing strategies. Round Robin routing is not implemented in JMVA, so it will be automatically transformed into Random, if you click the "Continue" button. Otherwise, you can change it following the link.
A state dependent routing strategy was found	Warning	You have defined a Routing strategy that doesn't match the one of the BCMP theorem. Click on the warning message and choose if you want to convert all non state independent routing strategies to Random routing or change the routing strategy manually.
ServerX with non valid service time distribution	Warning	You have defined a service time distribution not accepted by MVA. Modify the service time distribution to an exponential one. If "Continue" button is pressed, this distribution is considered as an Exponential with the same mean value of the original one.
What-If Analysis (WIA) not available	Warning	When you have defined a WIA and you want to export the model in JMVA, this error appears because the JMVA can't support WIA. If "Continue" button is pressed, WIA is ignored.
Different per-class queueing strategy found at StationX	Warning	This error appears during the export in JMVA because you have defined, for the stationX, a queue strategy different for the various classes. To export in JMVA, all queue strategies must be equal for all the classes. If "Continue" button is pressed, each queue strategy is considered as "Processor Sharing".
Non uniform service strategy inside FCFS station StationX	Warning	A few constraints apply as for the admissible service time distributions. A FCFS station must have exponentially distributed service time with the same mean for all the classes of customers that visit the station. If you click the "Continue" button, this issue is ignored.
Non exponential service time inside FCFS station StationX	Warning	A few constraints apply as for the admissible service time distributions. A FCFS station must have exponentially distributed service time with the same mean for all the classes of customers that visit the station. If you click the "Continue" button, this issue is ignored.
In StationX, rows must sum to a value equal to or greater than one	Error	This error appears because some rows of the class switch matrix of StationX do not sum to a value equal or greater than one.
StationX is followed by a queue with BAS strategy scheduling	Error	This error appears when StationX is a class switch station and it is directly connected with a queue implementing a BAS strategy scheduling. This topology should make performance indices computation inconsistent, thus it is forbidden.
StationX between fork/join. This topology is not allowed	Error	This error appears when StationX is a class switch station and is placed into a path that connects a fork and a join station. In this scenario is hard to define which is the class of the jobs at the exit of the join station, thus this topology is forbidden.

3.14 Simulation results

The results generated are function of the type of simulation performed: a *single* simulation run or a *What-If Analysis* run. Many factors affect the simulation results obtained with a model. Among them are: transient length, confidence intervals and max error requested, max number of samples, seed selected. The automatic *transient detection* algorithms implemented in JSIM are described in Sect.3.14.4.

3.14.1 Results from a single simulation run

At the end of a simulation, a panel with several tabs corresponding to the performance indices previously selected is shown. If there are several indexes of the same type, for example if the `Response Time` is referred by two or more stations or different classes, you will see the graphs under the same tab (see Figure 3.103).

In each window, results for all the stations for which the performance index is specified are listed. Numerical values and graphs are given. Successful results, i.e., when the confidence interval for the index has been computed with the precision set by the user, are indicated by the check-mark green symbol  (see Figure 3.103). In all the other cases, the red symbol indicates that the results obtained are *not* within the requested confidence interval.

In case of a syntactically correct network with some connections that make no sense from a modeling point of view, e.g., a station that is never visited by any customer, a yellow question mark  indicates that the simulator could not compute the index for lack of measurements.

Values are available for the following parameters:

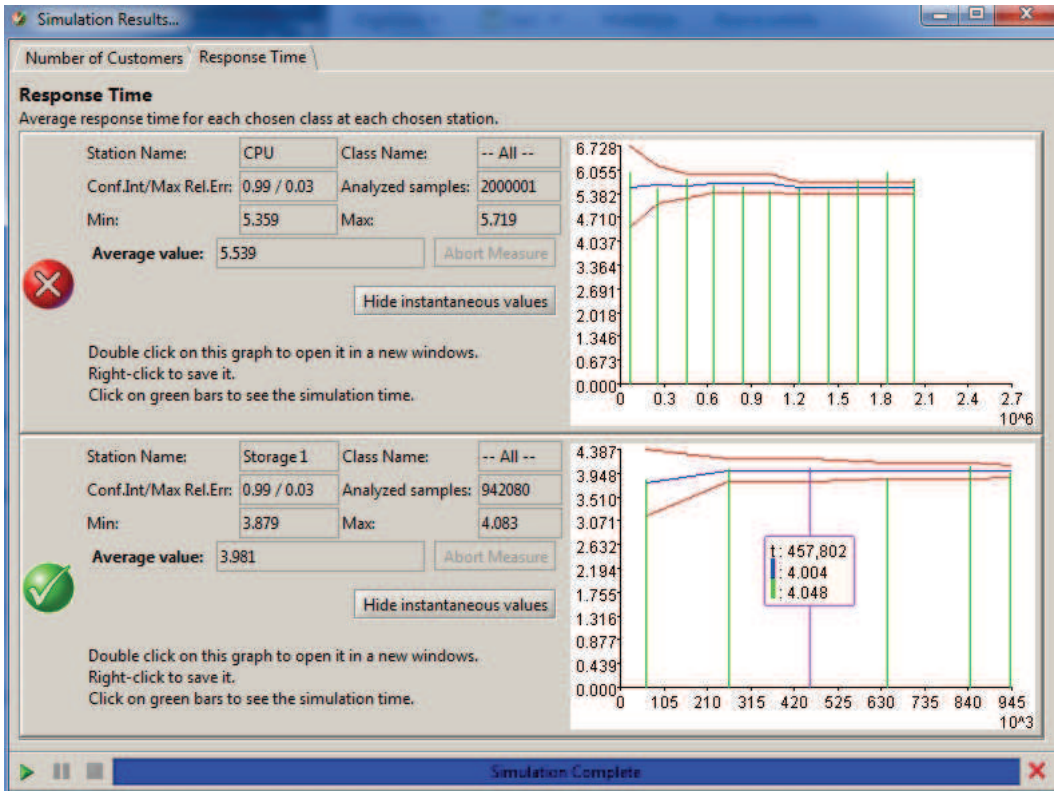


Figure 3.103: Results of a simulation run.

- `Station Name:` This is the name of the station for which the index is computed
- `Class Name:` This is the name of the class for which the index is computed at the station (it could be "All", to mean All Classes)
- `Conf.In / Max Rel.Err.:` This is the Confidence Interval and Maximum Relative Error of the computed index
- `Analyzed Samples:` This is the number of samples (the collected data) used to compute the performance index
- `Min.:` This is the minimum value observed for the index
- `Max.:` This is the maximum value observed for the index
- `Average Value:` This is the computed average value of the index, usually the value of greater interest.

Double-click on a graph (Figure 3.103) to magnify it. The x-axis represents the simulation time, i.e., the time considered by the simulator, that is event-driven. Every fixed number of detected values (a value set internally by default, see Sect.3.14.4), i.e., a *sample interval*, the simulator compute:
— the mean values of the index, connected with a *blue line*, computed on a number of measurements which increases with the progress of the simulation. All the collected values are considered at each computation except the ones discarded when detected as transients.
— the Confidence Intervals, *red lines*, of the values detected in the last set of values considered by the simulator,
— a *green bar* representing the mean value of the index computed with the measurements of the last interval. Clicking on a green bar a popup will appear showing the corresponding simulation time (t), the mean value (blue), and the instantaneous value (green).

The values are plotted in the graph when an *animation interval* expires (to set click on the icon from the toolbar, see Figure 3.35). Since the animation intervals, the intervals used to plot the values on the graphs, are based on the wall time while the sample intervals (computation intervals) are based on the simulation time (that is event-driven) the interval between consecutive bar charts may be irregular.

3.14.2 Result of What-If Analysis simulation run

At the end of a *What-If Analysis* simulation run, a panel showing the statistical values of the indexes, a plot of the indices behavior with respect to the control variable of the What-If Analysis and a table are shown. The panel shows a number of tabs corresponding to the number of indices inserted.

For each tab, i.e., each performance index, the results are grouped into three parts:

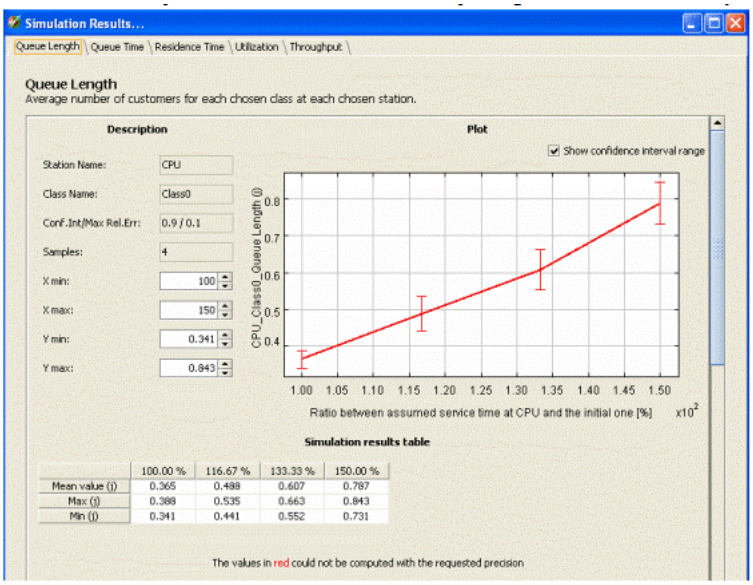


Figure 3.104: Results of a What-If-Analysis.

- The first part consists of the names of the index, of the station, and of the Class, the number of samples executed, the confidence interval and the maximum relative error, and the minimum and maximum values of the index.
- The second part is represented by a graph with a grid to facilitate the reading and the X and Y axes with their measure unit and scale. By default, on the graph are plotted the confidence interval range for all the executions with different values of the What-if analysis control variable, that can disabled through the check box.
- The third part consists of a table showing the results of each simulation run. The number of columns of the table corresponds to the number of steps executed. When the simulation engine cannot compute the confidence interval or the max relative error with the required probabilities, the values are written in red.

The size of each plot can be magnified double-clicking on it.

3.14.3 Export to JMVA

A simulation model created with JSIMgraph can be exported to the analytic solver component of the JMT suite, namely JMVA. In the `Action` menu, click `Export` to MVA.

If there are errors or conditions not allowed in models to be solved by analytical techniques, a dialog window with information about such errors appears. If you want to continue with the analytical solution, you must correct the errors.

Refer to JMVA user's guide for specific help on JMVA functionalities.

3.14.4 Transient detection

Simulation results are analyzed using transient detection and confidence interval estimation algorithms. These techniques are executed online, and can stop the simulation if all accuracy requirements are met. Confidence intervals are generated using the spectral method presented in [HW81]. The effectiveness of this technique depends on the stationarity of the sample distribution. If the series of samples shows trends or pronounced non-stationary behaviors, the estimate of its power spectrum can be unreliable, and this degrades the quality of the confidence intervals generated by the spectral method which works in the frequency domain. Therefore, techniques for discriminating if a group of samples is stationary are important to maximize accuracy, since detecting and removing non-stationary data can greatly improve the quality of power spectrum estimates. For this purpose, we have implemented transient detection using the R5 heuristic [Fis73] and the MSER-5 stationarity rule [Spr98]. The related transient detection and removal flowchart is shown in Fig.3.105. During the execution of the simulation, the statistical analyzer is fed with samples of each performance measure. First the R5 heuristic is evaluated until it indicates the termination of the transient. Afterwards, the MSER-5 is run

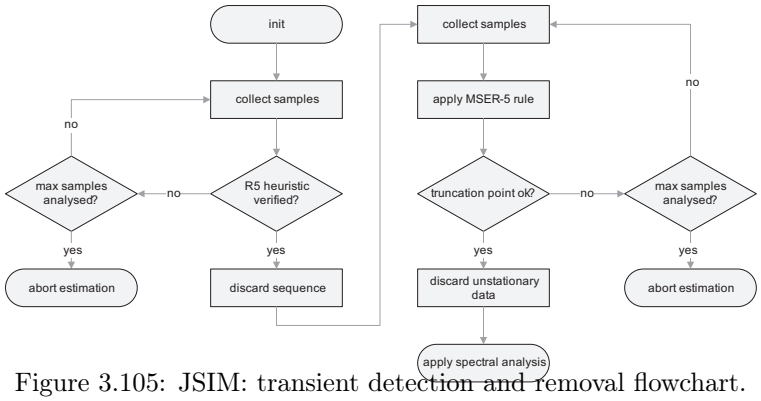


Figure 3.105: JSIM: transient detection and removal flowchart.

to double-check the indication of the R5 heuristic. When also the MSER-5 test indicates stationarity, the data is sent to the statistical analyzer for the final generation of confidence interval using the spectral method of [HW81]. A transient period is characterized in the simulation graph of a performance index by the absence of the red lines delimiting the confidence interval. In Fig.3.106 it is possible to see the transient period (from 0 to about 260 sec of clock time) for the index *System Response Time of class C1.*
We now review the R5, MSER-5 and spectral methods and outline their implementation in the JMT simulator.

The R5 heuristic control
The R5 heuristic [Fis73] detects the initial transient period by checking if the time series of samples crosses its mean value more than k times, where k is a user-specified parameter. We implemented this rule using the indications of Pawlikowski [Paw90]. In order to limit time and space requirements, we periodically check transient termination using a period of 500 samples. If the new samples do not satisfy the crossing condition, 500 new samples are collected before a new check is performed. After matching the stopping criteria, the $R5$ heuristic is complete and the simulator performs a second transient detection using the MSER-5 rule. This is done because we noted that applying several rules empirically reduces the number of wrong detections.

Concerning the choice of the k parameter, Gafarian et al. [GAM78] recommend $k = 25$ for $M/M/1/\infty$ models, while Wilson and Prisker [WP78] recommend $k = 7$ for $M/M/1/15$ models. We inspected more than 50 single workload simulation models with randomly generated networks including up to 4 load-independent and load-dependent queues, and checked the behavior of utilization, throughput, queue-length and response time metrics. We observed that the values proposed in the literature did not correctly igdentified transients on some of the instances. In particular, there exist cases in which $k = 7$ gives an early detection during an unfinished initial transient ramp. Conversely, with the $k = 25$ value, the transient of the utilization metric tends to be very slow, and in some cases even hundreds of thousands of samples are discarded before moving to the MSER-5 rule. Since the last problem is less severe than a wrong generation of results, we have chosen a value $k = 19$ to maintain the general properties of the recommendation in [GAM78], while reducing the number of cases where the transient is excessively slow.

The MSER-5 Rule control
MSER-5 rule is an identification method for the best truncation-point in a data sequence, first presented in [Spr98]. Initially, we used Schruben's test for this purpose [Sch82], but the results were not always satisfactory, as observed also in [WCS00]. Since the MSER-5 rule is not generally meant for online analyses, but instead works on a fixed data set, we adopted the approach of Robinson [Rob05] which consists in getting online new samples unless the truncation point is detected before the half of the analyzed sample set. Our implementation uses circular lists of 5000 batches (25000 samples), and has constant access and computation times. We observed that using structures with increased size had no benefit on accuracy, but imposed additional computational overheads. In the simulator, the MSER-5 rule is reapplied periodically until the detection of the optimal truncation-point. We set the period equal to the number of samples discarded by the R5 heuristic.

The spectral analysis
The spectral analysis of Heidelberger and Welch [HW81] is a stable and computationally efficient method for computing simulation confidence intervals using variable batch sizes and a fixed amount of memory. In the JMT simulator, the method is run on the stationary part of the sequence of samples. The spectral analysis is periodically run until the required confidence intervals are found. Nonetheless, it happens quite often that the portion of data immediately available after the MSER-5 rule is already sufficient to compute the required

confidence intervals.

The spectral analysis proceeds as follows. Given a covariance-stationary sequence of samples $X(1), \ldots, X(N)$ of the performance measure X, the spectral analysis obtains an estimate of the power spectrum $p(f)$ at the frequency $f = 0$. The value $p(0)/N$ is an unbiased estimator of the sample variance, and therefore can be used to generate confidence intervals on X also if the sequence of sample is correlated. The advantage of the technique with respect to the standard variance estimator

$$Var(X) \sim \frac{1}{N-1} \sum_{j=1}^{N} (X(j) - \bar{X})^2$$

is that the latter does not hold if the samples are correlated. It is known from long-time that performance metrics in queueing network models are correlated, see, e.g., [BKK84] for correlations in response times of product-form models, and this motivates the use of spectral analysis methods in the JMT simulator.

The spectral analysis requires to first compute the sample period gram $I(n/N)$, $0 < n < N/2$, and then generate the function $J(n) = \log(0.5(I((2n-1)/N) + I(2n/N)))$ which is the logarithm of the mean period gram over consecutive values of $I(n/N)$. The most important property of $J(n)$ is that its functional shape is linear in a neighborhood of the zero frequency, thus the value of $I(0)$ (which gives the power $p(0)$) can be estimated by regression of $J(n)$.

We implemented the regression technique using singular value decomposition, and fitting the data to a polynomial of order two. This implementation is consistent with the graphs in [HW81] which show that for batch sizes of 4-8 samples $J(n)$ is approximately linear or quadratic in a neighborhood of zero.

3.14.5 Dead performance index

In JSIM there is a control for detecting if no (or very few) values are detected for a performance index. This event may happens in models not well parameterized. For example, after an initial transient it may be that there are paths in which do not transit customers for a long time (because their routing probability is very low or other peculiarities of the routing policy selected), and hence the resources linked to them do not detect new values for the corresponding performance indices. We will refer to these indices as "dead indices".
The implemented test controls that, for each index, the number of samples received every 240000 increments of the simulation clock is different from zero. If it is zero, the collection of sample for that index is stopped and a *Orange question mark* is shown. In Fig.3.106 it is shown a dead index (System Response Time for class C2) detected. Note that the number of samples collected for this index is not zero (they are 3052), but clearly they were detected all in the transient phase since the average value shown is zero (the samples collected in a transient phase are not considered for the computation of the index value). If the average value of a dead index is different from zero means that the sample collection has been stopped after the end of the transient phase, that has been detected. The average value refers to the small number of samples detected.

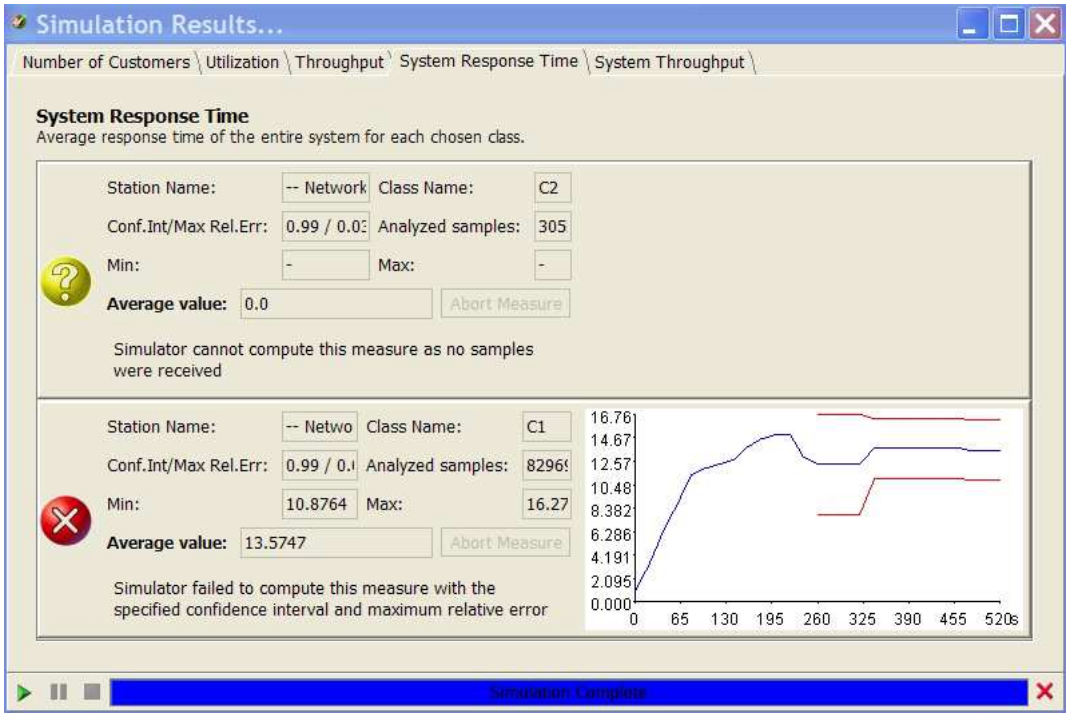


Figure 3.106: Detection of a *dead index*: System Response Time for class C2.

3.15 Statistics on performance indexes

In this section we will describe how the statistical values of the performance indexes generated as output of a simulation run can be obtained and how their values are computed.

3.15.1 How to obtain Statistics and Graphs of the simulation results

For each performance index selected it is possible to obtain several statistical values and graphs. To obtain the statistical values of an index you should check the box `Save Stats.` (*Statistical Results*) in the corresponding line of the `Define performance indices` window. For example, in Fig.3.107 the statistical values of the `Response Time` and of the `Utilization` have been required.
A CSV file with the data needed for the statistical computations is generated for *each* index selected. The default path of these files is C:\Users\`user name`\JMT\ and the default *names* are `Station name_index name.csv`, e.g., `Station 1_Response Time.csv`. You may change the path in the window shown in Fig.3.107. The structure of the files and the meaning of the data stored are described in Sect.3.15.2.

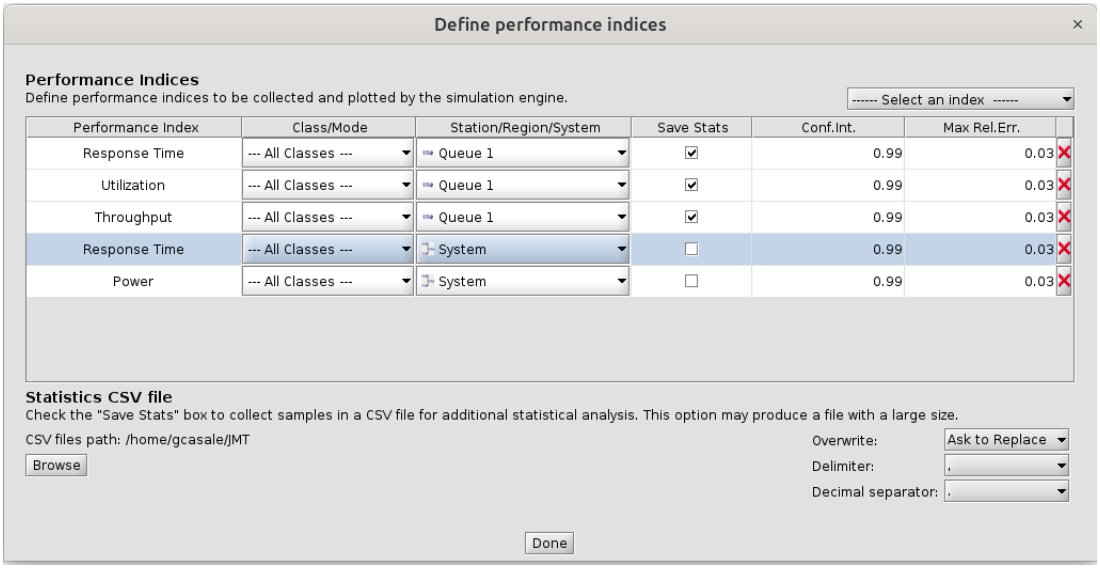


Figure 3.107: Selection of the `Response Time` and `Utilization` indexes for the collection of their values and the computation of their statistical metrics (see the `Save Stats.` check box).

At the end of the simulation run the `Statistical Results` button in the `Simulation Results` window of the index selected will be activated. By pressing on it, the window with the statistical values of the corresponding index (`Response Time` in this case) will be shown (see Fig.3.108).
The values considered can be filtered by the *number of the samples* collected (`First Sample` and `Last Sample`) or by the *simulation time* (`Initial Simulation Time` and `Final Simulation Time`) in the `Filter analyzed samples` boxes. A user may also filter the values considered in the graphs selecting the min and the max values in the boxes of the `Graph Drawing Options` section. The `Number of intervals` in which the values are grouped to draw the graphs can also be selected. Three types of graphs are available and can be selected in the `Graph Drawing Options` section of the window: `density line, histogram, distribution`. In Fig.3.109 the `Histogram` of the `Response Time` values is shown. The values computed for generating the graphs can be saved in CSV files.
When the `Distribution` type graph is selected, the values are sorted in increasing order. The *quartiles* divide the set of observations in 4 equal parts, and the *percentiles* in 100 equal parts. To plot the quartiles or the percentiles it is sufficient to set the graph type to `Distribution` and the `Number of intervals` to 4 or 100 respectively.

3.15.2 Computation of the statistical values

The computations required to obtain the statistical values of the various indexes are different as a function of the index considered. A *Statistical Results* file CSV is generated for *each* index selected, whose default path is C:\Users\`user name`\JMT\ and whose default *name* is `Station name_index name.csv`, e.g., `Station`

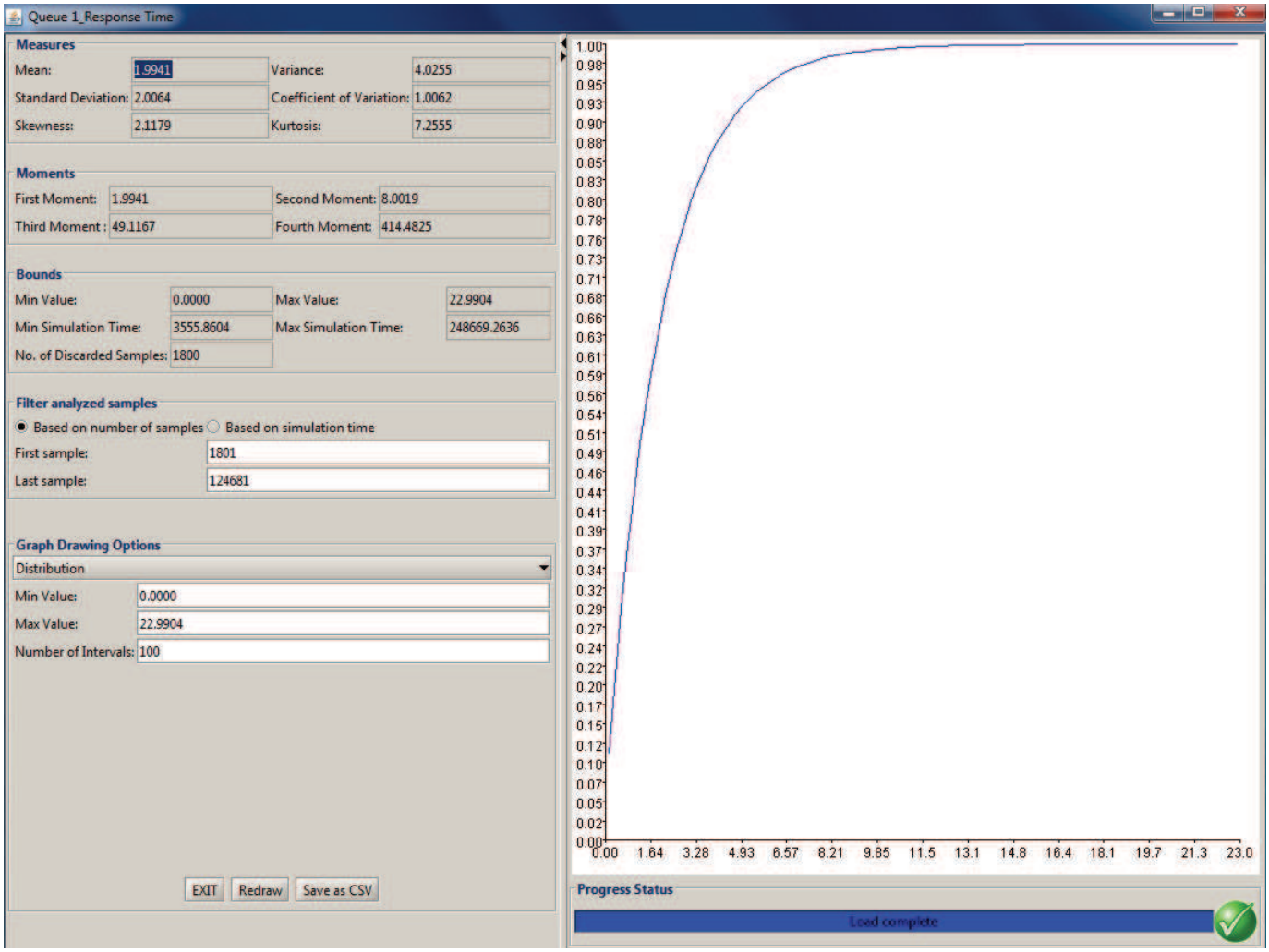


Figure 3.108: Statistical values of the `Response Time`, the graph selected is the *distribution*.

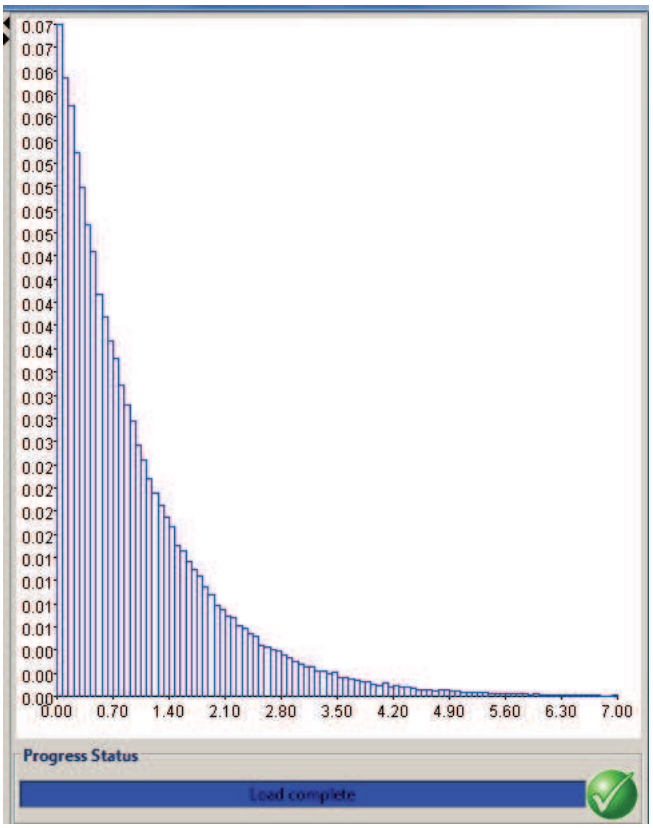


Figure 3.109: Histogram of the `Response Time` obtained grouping the values in 100 intervals (selection of the `Number of Intervals` parameter to 100) in the graph. The values represented are the *percentiles*.

`1_Response Time.csv`.
In Table 3.2 a synthetic description of the variables collected for each metric is shown.

The metrics can be subdivided into *two groups* as a function of the values of WEIGHT field used to compute the mean value of the sample collected. A *first group* is the one with WEIGHT=1.
Let us consider, for example, the `System Response Time` index of a simulation run in which n *jobs* where executed. When job i exit the model, a row of the *Statistics csv file* shown in Fig.3.110 is written. For this index, the values of data fields are:
(1) `TIMESTAMP`: simulation time at which job i exits the model,
(2) `SAMPLE`: the `Response Time` R_i of the job i computed as the difference from the execution start time and its completion time,
(3) `WEIGHT`: a weight value W_i that will be used for the computation of the mean value of the index considered. For the indexes having the weight values W_i equal to 1 each line of the file corresponds to a single job and thus each element has the same weight for the computation of the mean value. Typically, the mean `System Response Time` of n jobs is obtained as:

$$R = \frac{\sum_{i=1}^{n} R_i}{n} \tag{3.5}$$

	A	B	C	D
1	TIMESTAMP	SAMPLE	WEIGHT	
2	1,481281719	0,28281	1	
3	2,112621364	0,584314	1	
4	2,760347034	0,534165	1	
5	3,142743561	0,786578	1	
6	3,37834445	0,963454	1	
7	3,682782122	1,211253	1	

Figure 3.110: Example of the Excel file `Queue 1_Response Time.csv` for the `Response Time` index of station `Queue 1`.

A *second group* of indexes is that for which the WEIGHTS are no longer constants but assume the values of *time-intervals* between consecutive events. Let us consider, for example, the index *mean* `Number of customers`

Performance Index	*TIMESTAMP* (event detected)	*SAMPLE*	*WEIGHT*
Response time (station) [s]	exit from station	time spent in a visit (*queue + service)*	1 (req)
System Response Time [s] (*referred to as R*)	exit from system (*from Ref.Station)*	time in the system (*queue + service)*	1 (job)
Residence time (stat.) [s]	exit from system (*from Ref.Station)*	time spent in station (*for a complete exec)*	1 (job)
Queue time (station) [s]	exit from queue (*enter service)*	time in queue (*waiting for service)*	1(req)
Throughput (station) [req/s]	exit from station	inter-departure time	1 (req)
System Throughput [j/s] (*referred to as X*)	exit from system (*from Ref.Station)*	inter-departure time	1 (job)
Drop rate (station) [req/s]	exit from queue (*dropped)*	time between drops	1(req)
System Drop rate [j/s]	exit from system (*dropped)*	time between drops	1(req)
System power [X/R]	exit from system (*from Ref.Station)*	X/R	1 (job)
No.customers (stat.) [req]	arrival/exit (*to/from station)*	no. of req. in station (*before the event)*	elapsed time from this event and the previous one
System No.of custom. [j]	arrival/exit (*to/from system)*	no. of req. in system (*before the event)*	elapsed time from this event and the previous one
Utilization (station) [%]	exit from queue (*the server is idle)*	0	elapsed time from this event and the previous one
Utilization (station) [%]	exit from server (*server was busy)*	1	elapsed time from this event and the previous one

Table 3.2: Performance indexes and related data stored in the corresponding files for their statistical processing.

of a station. When the request i enters/exits the station, a row of the *Statistics CSV file* shown in Fig.3.111 is written. The meanings of the data fields are:
(1) TIMESTAMP: simulation time at which request i enters/exits the station,
(2) SAMPLE: is the number m of requests that are in the station (in queue and in service) when request i exits the station,
(3) WEIGHT: a weight value W_i that will be used for the computation of the mean value of the number of requests. Each value W_i represents the duration of the interval of time in which the m requests of the SAMPLE that are in the station when the event is detected have been there. For the request i, the W_i value is computed as the difference $[TimeStamp\,(i) - TimeStamp\,(i-1)]$.
Let T_m be the time interval during which m requests were in the station, and max be the maximum number of requests that were in the station during the simulation run. The total simulation time T is given by $T = \sum_{m=0}^{max} T_m$. Typically, the *mean number N of requests* in the station for a simulation run of duration T is given by:

$$N = \sum_{m=1}^{max} \frac{mT_m}{\sum_{j=0}^{max} T_j} = \sum_{m=1}^{max} \frac{mT_m}{T} \tag{3.6}$$

JMT utilizes the following ***single formula*** to process **all** the indexes:

$$I = \frac{\sum_{i=1}^{n} S_i W_i}{\sum_{i=1}^{n} W_i} \tag{3.7}$$

where the W_i are the WEIGHTS values and the S_i are the SAMPLE values shown in Table3.2.

When the mean value of a index with all the W_i equal to 1 is to be computed, e.g., the Response Time, eq.3.7 becomes (in this case the S_i are equal to R_i):

$$I = \frac{\sum_{i=1}^{n} S_i W_i}{\sum_{i=1}^{n} W_i} = \frac{\sum_{i=1}^{n} R_i}{n} = R \tag{3.8}$$

	A	B	C	D
1	TIMESTAMP	SAMPLE	WEIGHT	
2	1,198471466	0	1,198471	
3	1,481281719	1	0,28281	
4	1,528307117	0	0,047025	
5	2,112621364	1	0,584314	
6	2,226181647	0	0,11356	
7	2,356165391	1	0,129984	
8	2,414890076	2	0,058725	
9	2,471529278	3	0,056639	
10	2,760347034	4	0,288818	
11	2,885456167	3	0,125109	

Figure 3.111: Example of data stored in the Excel file `Queue 1_Number of Customers.csv` for the `Number of Customers` index of station `Queue 1`.

providing the same result of eq.3.5.

Otherwise, when the mean of an index of the other type, e.g., the `Number of customers`, is to be computed, the W_i represent the time in which m requests were in the station, and the S_i values are the number of m requests in the station when the $i-th$ request leave the station. In this case eq.3.7 becomes:

$$I = \frac{\sum_{i=1}^{n} S_i W_i}{\sum_{i=1}^{n} W_i} = \frac{\sum_{i=1}^{n} S_i W_i}{\sum_{i=1}^{n} TimeStamp\,(i) - TimeStamp\,(i-1)} = \frac{\sum_{i=1}^{max} mT_m}{\sum_{i=0}^{n} T_n} = N \tag{3.9}$$

providing the same result of eq.3.6.

The indexes `Throughput, System Throughput, Drop Rate, System Drop Rate` obtained processing directly their values of SAMPLE and WEIGHT of the corresponding files are the inverse of the correct values. Indeed, the SAMPLE values are the inter-exit times of the requests/jobs. Thus, it became necessary to reverse the values obtained from their computations to show the correct statistics and graphs.

Let us remark that, as a function of the simulation condition, sometimes in the files of statistical values of some index (e.g., `Utilization, System Number of Customers`) it may appear a few lines with SAMPLE=WEIGHT=0 or only WEIGHT=0 and the *same* TIMESTAMP of the previous line. This is due to the working logic of simulator but of course do not affect the calculations of the statistical indexes.

3.16 Examples

Two simple examples of performance evaluation studies executed using JSIM*graph* are shown. We believe that readers could benefit from their analysis by concentrating on the abstraction process and on the model design rather than on the actual quantitative results presented in the projects.
The first example is a single simulation run whereas the second one is a `What-If` Analysis.

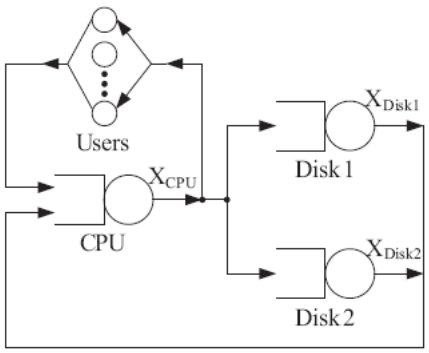


Figure 3.112: Layout of the system of Example 1.

3.16.1 Example 1 - A model with single closed class

For teaching purposes we are considering here the same *Example 1* presented in the JMVA user manual. We will see that the results obtained with JSIM*graph* include in their confidence intervals the correct values obtained

	CPU	Disk1	Disk2	Users
Service time [sec]	0.006	0.038	0.030	16
Visits	101	60	40	1

Table 3.3: The parameters for the single class model of Example 1.

with the exact solver JMVA.

The model
The digital infrastructure of a company consists of a computing server and a storage server equipped with two large disks arrays. The user requirements are homogeneous and the workload generated may be considered as a single class. The layout of the digital infrastructure is shown in Figure 3.112. The customer class, referred to as *ClosedClass*, has a population of $N = 3$ customers. There are four stations (*CPU, Disk1 and Disk2*), three are of queueing type load independent and one is of delay type (*Users*). Users delay station represents user's think time ($Z = 16s$) between interaction with the system. Service times, distributed exponentially, and the visits at the stations are reported in Table 3.3.

Drawing the graph

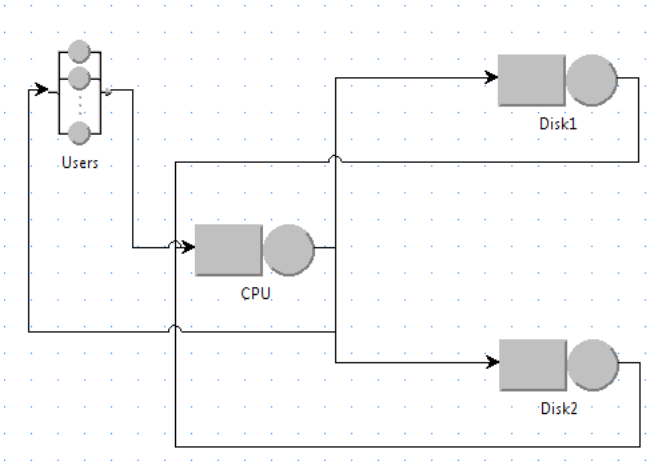


Figure 3.113: The system represented with JSIMgraph

- Select `File -> New` or click on the icon from the first toolbar.
- Insert a `Queueing station` in the drawing window by at first clicking the icon and then clicking on the drawing window. Perform this task 3 times in order to add 3 stations.
- Double click on a `Queueing station`. You will find a textbox to modify `Station Name`. By clicking each of the queueing stations, rename `Station Names` as *CPU, Disk1,* and *Disk2.*
- Insert a `Delay Station` by selecting the icon from the toolbar. Double click on it and rename the `Station Name` as *Users.*
- By selecting the icon, connect the queueing stations and the delay station as given in the problem. The graph should look like Figure 3.113.

Now we have to describe the values of the parameters.

Defining customer class

- Select `Define -> Customer Classes` from the menu bar or click on the class icon.
- Click on `Add Class`. Set `Population` as 3 and `Reference Station` as *Users.* Note that that `reference station` is the station which initiates and completes the flow of customers in the network.
- Click `Done` on the window of Figure 3.115.

Setting the values of the parameters

- Double click on `Users` delay station.

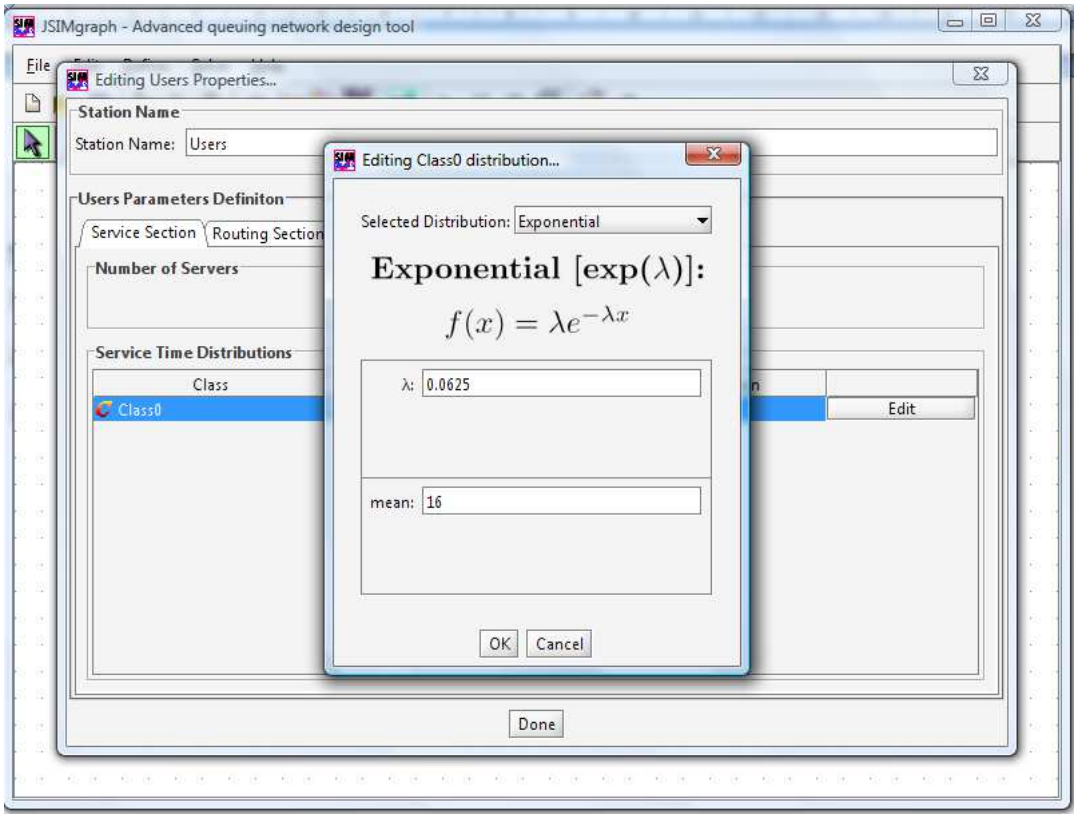


Figure 3.114: Setting the mean value of *Users Think time* for the delay station

- Click on `Edit`.
- The default `Selected Distribution` should be `Exponential` (see Fig.3.114). Set `Mean` value as 16 (recall that the Customer think time is Z=16s). Click `OK`. Note that the value of λ is automatically computed by the tool.
- Double click on *CPU*. Click on Service Section tab. Click on Edit.
- The default `Selected Distribution` should be `Exponential`. Set `Mean` value as 0.006. Click `OK`. Note that the value of λ is automatically computed by the tool.
- Click on the `Routing Section` tab. Set `Routing Strategies` as `Probabilities`. As there are 3 outgoing links from *CPU*, we have to set a job's probability to choose each of these paths. This is done by setting the probabilities of *Disk1, Disk2* and *Users* as 60, 40 and 1 respectively (see Figure 3.116). Click on `Done`. Note that the values of the `Probabilities` are automatically adjusted by the tool.
- Double click on *Disk1*. Click on `Service Section` tab. Click on `Edit`.
- The default `Selected Distribution` should be `Exponential`. Set `Mean` value as 0.038s. Click OK. The value λ is automatically computed by the tool.
- Click on the `Routing Section` tab. Choose `Routing Strategies` as `Probabilities`. Since there is only one outgoing link, which goes to the *CPU*, from *Disk1* so select the `Probability` of going to CPU (under the Routing Options subsection) as 1 (see Figure 3.117). Click on `Done`.
- Double click on *Disk2*. Click on Service Section tab. Click on Edit.
- The default `Selected Distribution` should be `Exponential`. Set `Mean` value as 0.030. Click OK. The value of λ is automatically computed by the tool.
- Click on the `Routing Section` tab. Choose `Routing Strategies` as `Probabilities`. Since there is only one outgoing link, which goes to *CPU* from *Disk2*, set the `Probability` of going to CPU (in the `Routing Options` subsection) to 1. Click on `Done`.

Defining the Performance Indices

Before start the simulation we must select the performance indices to analyze. In Example 1 we want to study the following indices:

– *Queue length [jobs]* (for CPU, Disk1, Disk2, Users and the global System)
– *Throughput [jobs/sec]* (for CPU, Disk1, Disk2, Users and the System)
– *Residence time [sec]* (for CPU, Disk1, Disk2, Users and the System)

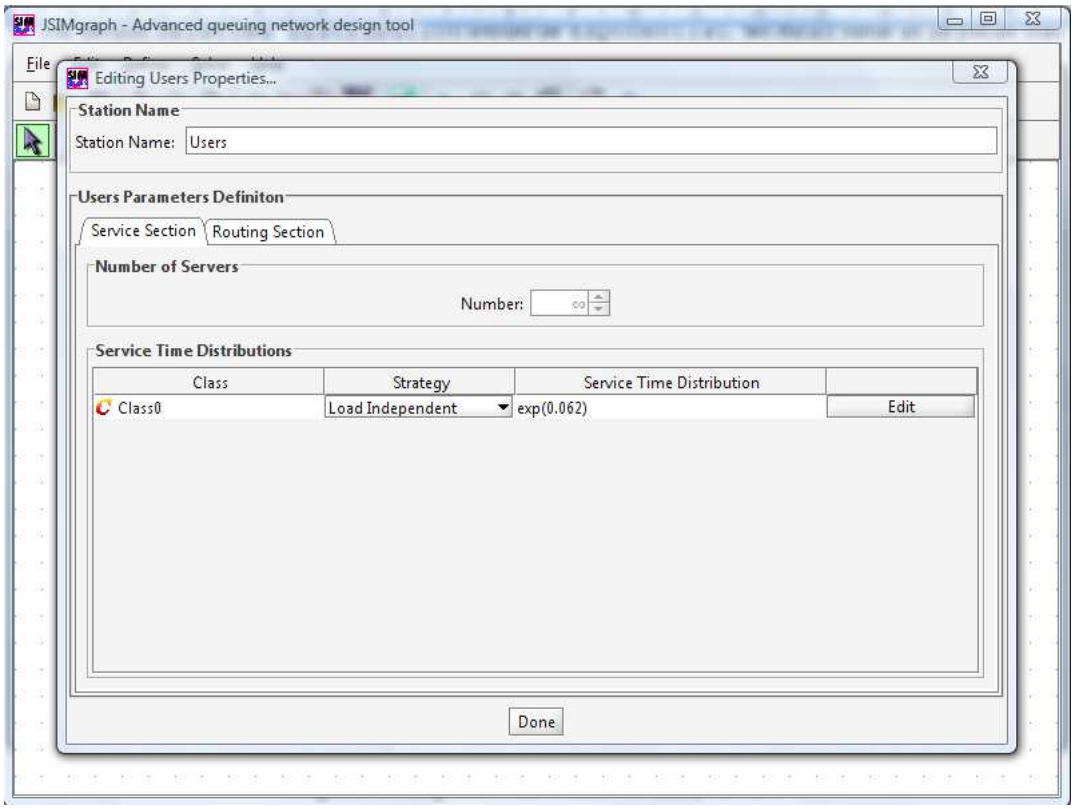


Figure 3.115: The Users properties window

– *Utilization* (for CPU, Disk1, Disk2 and Users)

The following actions should be done:

- From menu bar select `Define -> Performance Indices` or select from the toolbar.
- From the drop-down menu at the right, select Queue Length. Click on `Add Selected Index` for four times. Change `State/Region` of these Indices to *CPU, Disk1, Disk2* and *Users*.
- From the drop-down menu at the right, select `Throughput`. Click on `Add Selected Index` for four times. Change `State/Region` of these `Indices` to *CPU, Disk1, Disk2* and *Users*.
- From the drop-down menu at the right, select `Residence Time`. Click on `Add Selected Index` for four times. Change `State/Region` of these `Indices` to *CPU, Disk1, Disk2* and *Users*.
- From the drop-down menu at the right, select `Utilization`. Click on Add `Selected Index` for four times. Change `State/Region` of these `Indices` to *CPU, Disk1, Disk2* and *Users*.

The `Define Performance Indices` window should look like Figure 3.119. It should be noted that we can change the `Confidence Interval` and/or `Maximum Relative Error` for any performance index from this window. At the end click on `Done`.

Running the Simulation

To run the simulator - press `Alt+s` or select `Simulate` from the `Solve` menu or press from the toolbar. Note that if you had run the simulation for the same program before, JSIM*graph* asks for a confirmation whether you want to replace the previous simulation results or not. Choose `yes` to confirm. In the drawing window of the Simulator (Figure 3.120), the colored parts of a station represents the percentages of station busy time and of the queue length due to the customers of *Class0*. In the `Simulation Results` window, Figure 3.121, opened automatically when simulation starts, the progress of the average values of the selected performance indices and the extremes of the confidence intervals are shown.
Table 3.4 shows the average, min and max values of the indexes at the station level at the end of the simulation. Table 3.5 shows the average, min, and max values of Response Time, Throughput, and number of customers at the level of the global system.
Note that the results of different simulation runs may slightly vary due to the differences in the sequence of random numbers generated with different seeds.

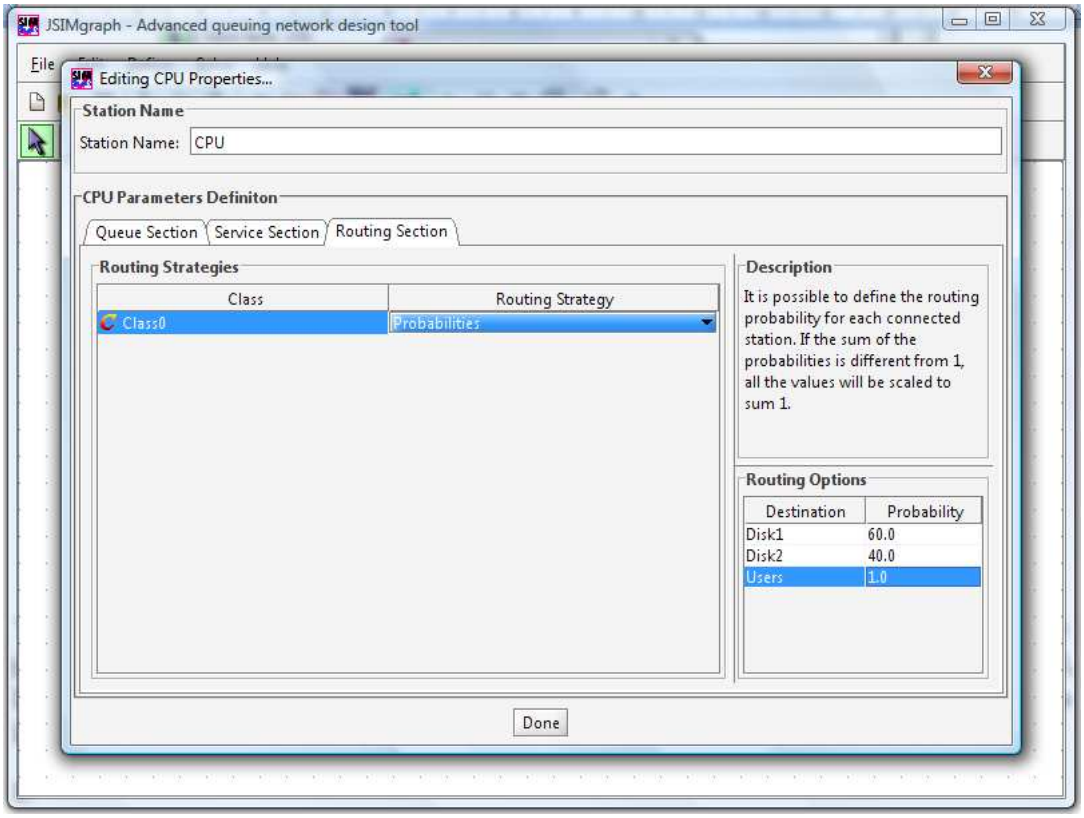


Figure 3.116: Setting the probabilities (or the visits) in output of *CPU* station

3.16.2 Example 2 - What-If analysis of a system with multiclass customers

The problem

Consider an intranet consisting of a storage server and an Apache server. The storage server consists of three disks used according to a RAID0 algorithm (striping). The web server consists of two independent systems that for performance issue can accept globally a limited number of customers in concurrent execution, i.e., they are allocated in a Finite Capacity Region with the maximum number of customers *MaxClients = 100*. A *load balancer* split the requests among them according to policies described later.

The customers are subdivided into two *open* classes, *Class0* and *Class1*: the first class access only the Apache server while the customers of the second class visit either the storage or the Apache server according to a

		Queue length [jobs]	Residence time [sec]	Utilization [%]	Throughput [jobs/sec]
CPU	min	0.080	0.882	0.076	13.067
	max	0.096	1.019	0.089	15.014
	avg	0.088	0.950	0.082	13.973
Users (delay)	min	2.157	15.681	2.157	7.496
	max	2.463	17.302	2.463	8.606
	avg	2.310	16.492	2.310	8.013
Disk1	min	0.373	2.836	0.276	4.996
	max	0.434	3.327	0.318	5.713
	avg	0.403	3.082	0.297	5.331
Disk2	min	0.181	1.326	0.159	0.135
	max	0.216	1.495	0.185	0.149
	avg	0.199	1.410	0.172	0.141

Table 3.4: The results of the simulation at the stations level.

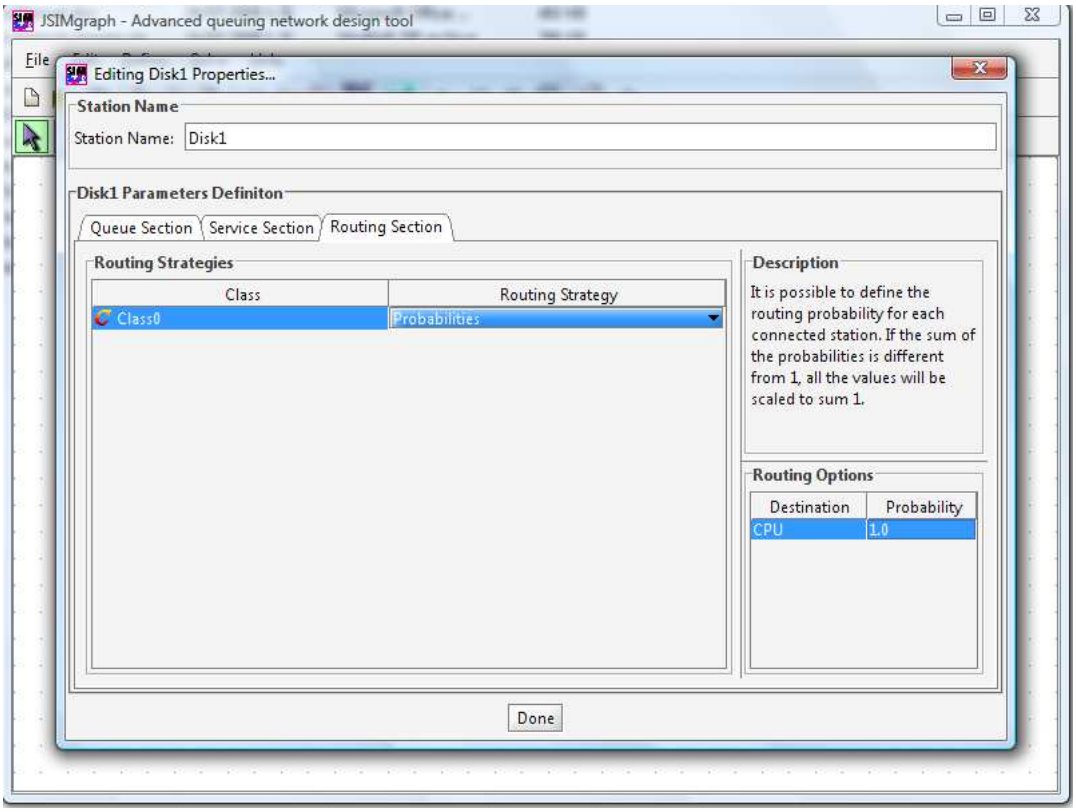


Figure 3.117: Setting the routing probabilities (or the visits) in output of *Disk1* station

	System Response Time [sec]	System Throughput [job/sec]	Customers number [jobs]
Min	20.510	0.134	2.952
Max	21.413	0.149	3.047
Avg	20.961	0.141	3.000

Table 3.5: The results of the simulation at the system level.

probabilistic algorithm. All *Class0* customers go to the *Load Balancer* and their interarrival times are hyper-exponentially distributed with parameters $(p, \lambda_1, \lambda_2) = (0.291, 0.182, 0.442)$, see Figure 3.123, resulting in a mean value equal to 3.2 sec and coefficient of variation equal to 1.192.
The interarrival times of Class1 customers are exponentially distributed with mean 5 sec. In order to model the RAID0 striping algorithm a Fork station is used. When a request arrive at the Fork station RAID0 it is splitted in three requests sent in parallel to all the disks of the storage server. Fork degree at *RAID0* station is 3, that is, for each incoming request three requests are generated in each outgoing link. Service times for all the disks - *Disk1, Disk2* and *Disk3* are 5 for Class0 and 0.5 for *Class1* customers respectively. An FCR is used in order to model the Apache server, which contains two web servers - *Web Server 1* and *Web Server 2*, that accept a maximum number of requests equal to 100. The *load balancer* station in this FCR split the load among the two web servers with the following algorithms: *Class0* customers with `Join the Shortest Queue` and *Class1* customers with `Shortest Response Time`. The service times for both Web Servers are *1 sec* for both the classes.
According to some enterprise objectives, we need to study the behavior of this system when the `Service Times for all classes` of *WebServer1* increase from current values to 150%. Particularly, we want to forecast the `System Response Time` for both *Class0* and *Class1* customers at *Confidence Interval* 0.99 and *Maximum Relative Error* 0.05. We want also to know the global `throughput` of *WebServer1* at *Confidence Interval* 0.90 and *Maximum Relative Error* 0.10. Thus, the control parameter that should be selected in the What-If Analysis is the `Service Times for all classes`.

Defining the model

- Insert *Source, Fork, Join, Routing, Queueing Stations* and *Sink* as required in the problem considered. Then connect them according to the specifications.

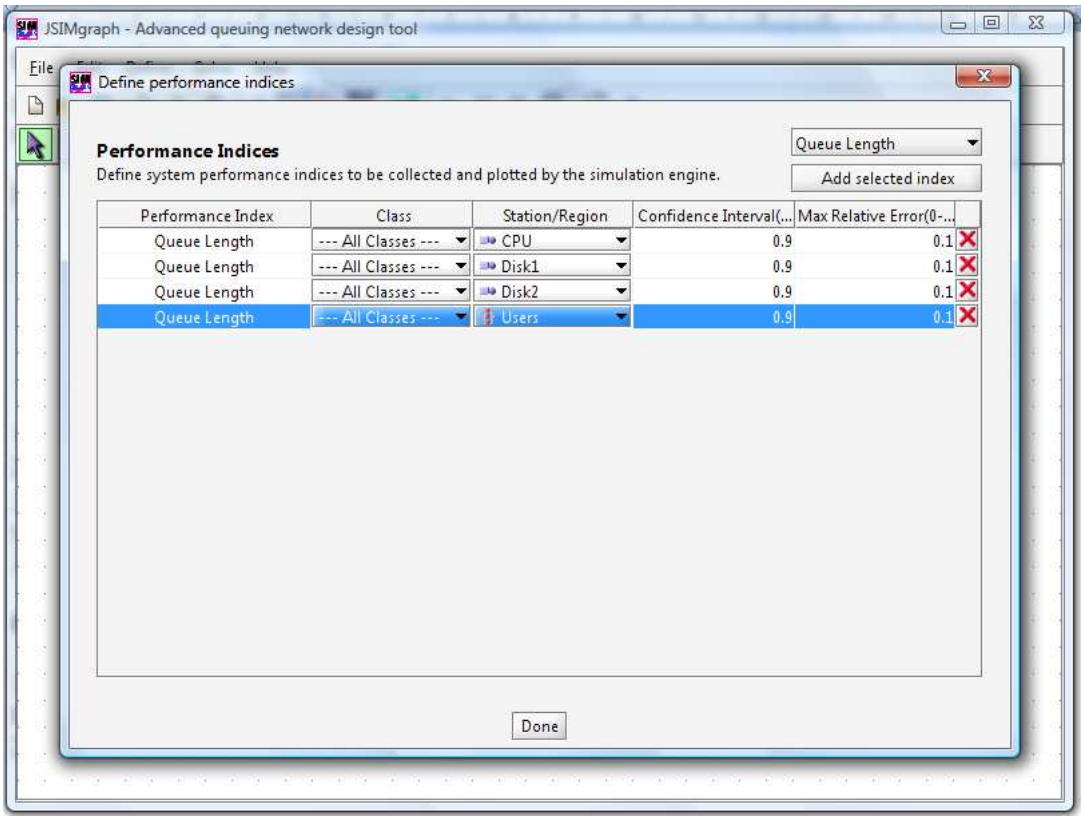


Figure 3.118: Setting the `Queue length` performance index

- Double click on the added resource, rename each of them as given in the problem.
- Hold Ctrl key on the keyboard and select `Load Balancer`, `Web Server 1` and `Web Server 2`. Then click on the  icon. It will create a new Finite Capacity Region consisting of those three elements.
- Double click on the finite capacity region, rename it as Apache, uncheck the infinite checkbox and set Region Capacity as 100. Click `Done`.
- From the menu bar choose `Define -> Customer Classes`. Add two open customer classes - *Class0* and *Class1*. *Class0* is hyperexponentially distributed with $(p, \lambda_1, \lambda_2) = (0.291, 0.182, 0.442)$. *Class1* is exponentially distributed with mean value 5.
- Double click on `Requests`. Set `routing strategy` for both *Class0* and *Class1* as *Probabilities*. Since all *Class0* customers go to the `LoadBalancer`, set `Probability` of the `LoadBalancer` as 1. Whereas, since the *Class0* customers may go either to the `LoadBalancer` or to the RAID0 station, set the `Probability` of both `LoadBalancer` and RAID0 as 0.5. Then Click on `Done` (see Figure 3.124).
- Double click on RAID0 and set `Fork degree` at 3 (see Figure 3.125).
- Double click on the disks, then, clicking `Edit` in the `Service Section` tab, set the mean values for all the disks - *Disk1, Disk2* and *Disk3* as 5 for *Class0* and 0.5 for *Class1*.
- Double click on the `LoadBalancer`. Set the `Routing Strategy` of *Class0* to `Shortest Queue length`, and that of *Class1* to `Shortest Response Time` (see Figure 3.126).
- By double clicking on the Web Servers, then under the `Service Section` tab, by clicking `Edit`, set the mean values for all the web servers - *WebServer1* and *WebServer2* as 1 for both the classes.
- From the menu bar `Define -> Performance Indices` add the index `System Response Time` twice - the first one for *Class0* and the second one for *Class1*. Change the `Confidence Interval` to 0.99 and `Maximum Relative Error` to 0.05. Then add the index `Throughput`. Set its `Station/Region` as *WebServer1* (see Figure 3.127).
- From the menu bar choose `Define -> What-if analysis`. Check the checkbox `Enable What-if analysis`. Select the `Service Times` parameter. Set `To(%)` as 150, `Steps (no. of exec.)` as 10 and `Station` as *WebServer1*. Keep the values of the `Arrival Rates` parameter as it is (`To` should be 150 and `Steps` should be 10). Steps in the Seed parameter should remain 10 by default. Click `Done` (see Figure 3.128).
- To run the simulator - press `Alt+s` or select `Simulate` from the `Solve` menu or press  from the toolbar.

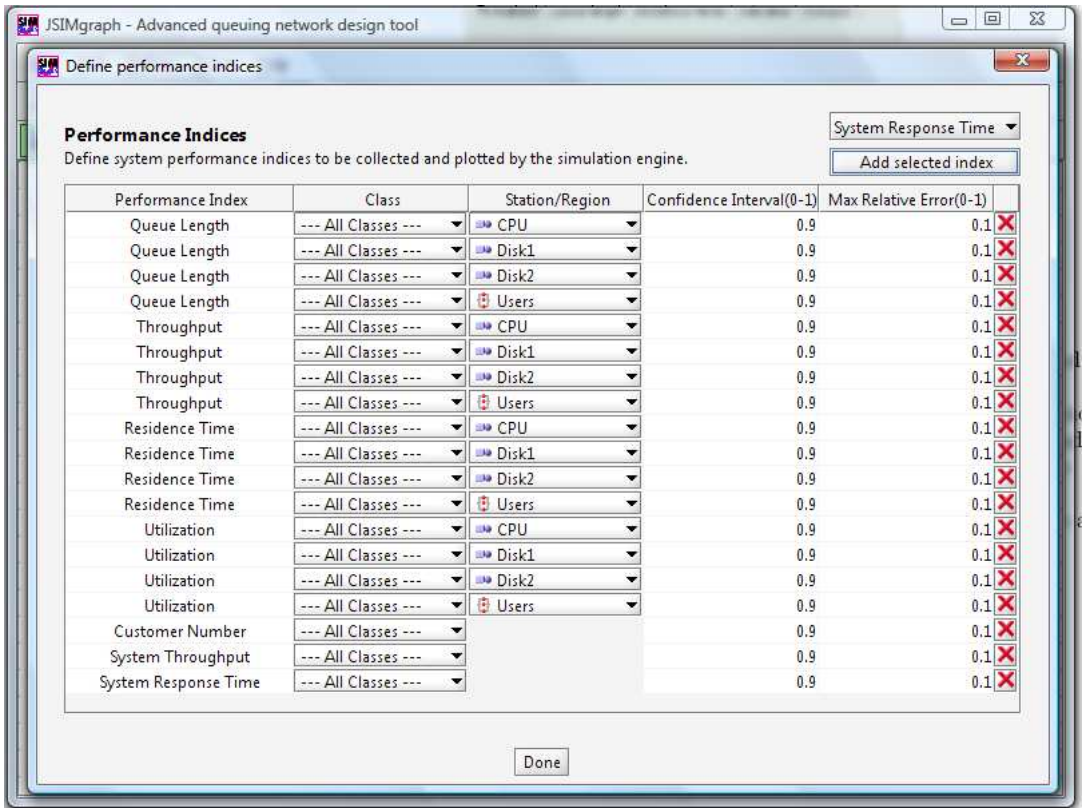


Figure 3.119: The `Define Performance Indices` window at the end of the selection of the indices

Note that if you had run the simulation for the same program before, JSIM*graph* asks for your confirmation whether you want to replace the previous simulation results or not. Choose `yes` to carry on. If we look the drawing window of the Simulator, we can see the simulation going on.

The windows of Fig.s 3.129, 3.130, 3.131 show the results of the What-If Analysis. Note that the results of different simulation runs may slightly vary due to the differences in the sequence of random numbers generated with different seeds.

3.16.3 Example 3 - Using FCR for modelling bandwidth contention

The FCR (Finite Capacity Region) can be used to model the contention of resources that can be located in several positions in the model even if they are *not adjacent*. This feature can be particularly useful if we have to model the contention of bandwidth. The queue for entering the FCR accept priorities, so a request with high priority may enter the FCR as soon as one request exit the FCR (when the number of requests inside the FCR is equal to the max admitted).

The problem definition
Consider a configuration consisting of a workstation sending requests over an Intranet which is composed of Web server, Application Server and Database Server and can handle only up to 100 requests at any given time. The application server in the Intranet further connects itself to a remote server using Ethernet connection which clearly can handle only 1 request at any time.

How to model

- Insert Source, Routing Station, Queueing Stations and Sink.
- Double click on the added resources and rename them as shown in the Figure 3.132.
- Hold the Ctrl and select the Web Servers, Application Server and Database Server and then click the  icon. It will create the first Finite Capacity Region.
- Double click on the Finite Capacity Region and rename it to `Intranet`, uncheck the infinite checkbox and enter a region capacity (max = 100) as shown in Figure 3.133.

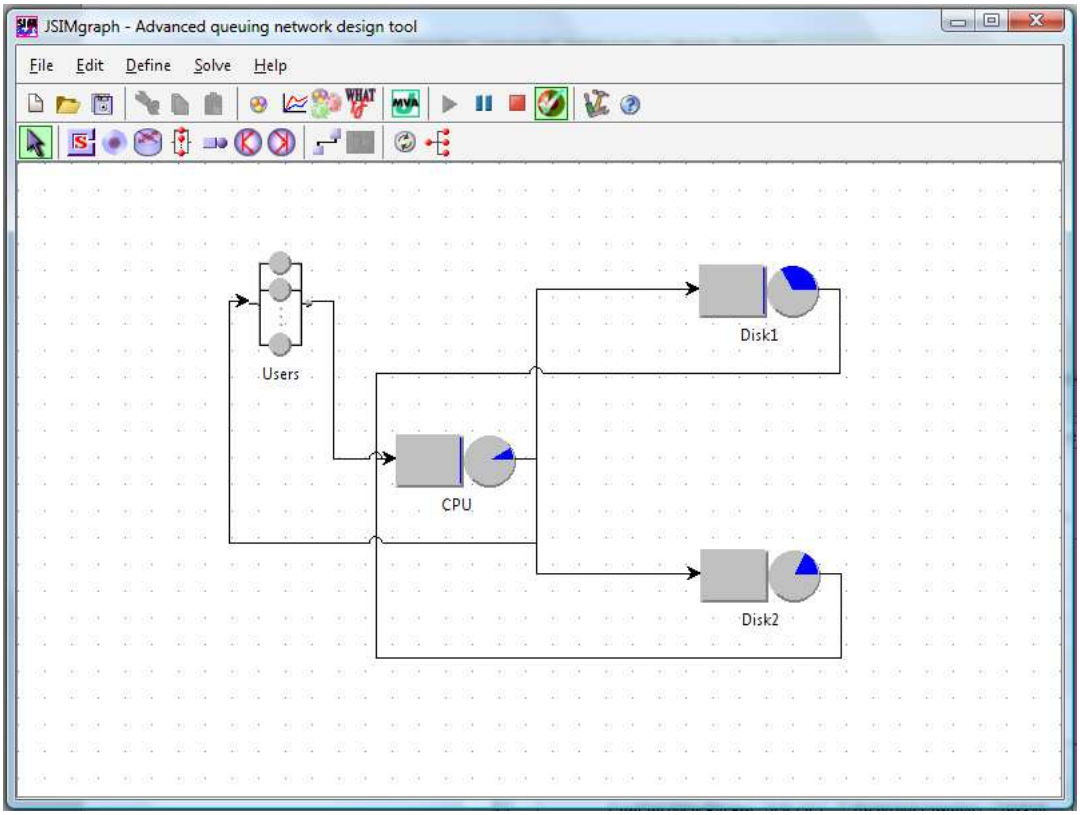


Figure 3.120: Example 1 - The layout of the model with the percentage of utilization and queue length (colored areas)

- Add the classe(s) from the menu.
- Now hold the Ctrl and select the transmission system and Ethernet Cable and click icon. It will create another Finite Capacity Region which represents data flow over a Ethernet cable.
- Double click on FCR and rename it to `Ethernet`, uncheck the infinite checkbox and set the max region capacity to 1.
- Insert the Performance indices you are interested to evaluate.
- Insert the preferred routing and queueing policies.
- Run the simulation.

It is an interesting topic of study in performance evaluation is to model the contention for a set of resources, that can be either directly connected or not connected at all. As seen above the FCR structure allows to fix the maximum number of customers in a set of resources and also allows the priority scheduling.

3.16.4 Example 4 - Closed models with class-switch

Class-switch allows the definition of models where customers may change class (i.e., their characteristics) during the execution.
In such models the number of jobs of any class may no longer remain constant, thus the *population mix* vector is not constant (for a formal definition of the *population mix* vector please refer to section 6.2.6).
In this example we run a model with an initial population mix and analyze what is the equilibrium state of the population mix vector.
Open JSIM*Graph*, click on and draw the model reported in Figure 3.134. As mentioned before we want to study how the population mix vector evolves over the time, thus first of all we have to set the initial state.
Click on class and, as shown in Figure 3.135, add two closed classes and set their *Population* field to 500.
In this way the initial population mix vector is $[0.5, 0.5]$. In order to set the $\mathbf{C}$ matrix of the class switch component double click on it and input the following matrix (Figure 3.136)

$$\mathbf{C} = \begin{vmatrix} 0.8 & 0.2 \\ 0.6 & 0.4 \end{vmatrix} \tag{3.10}$$

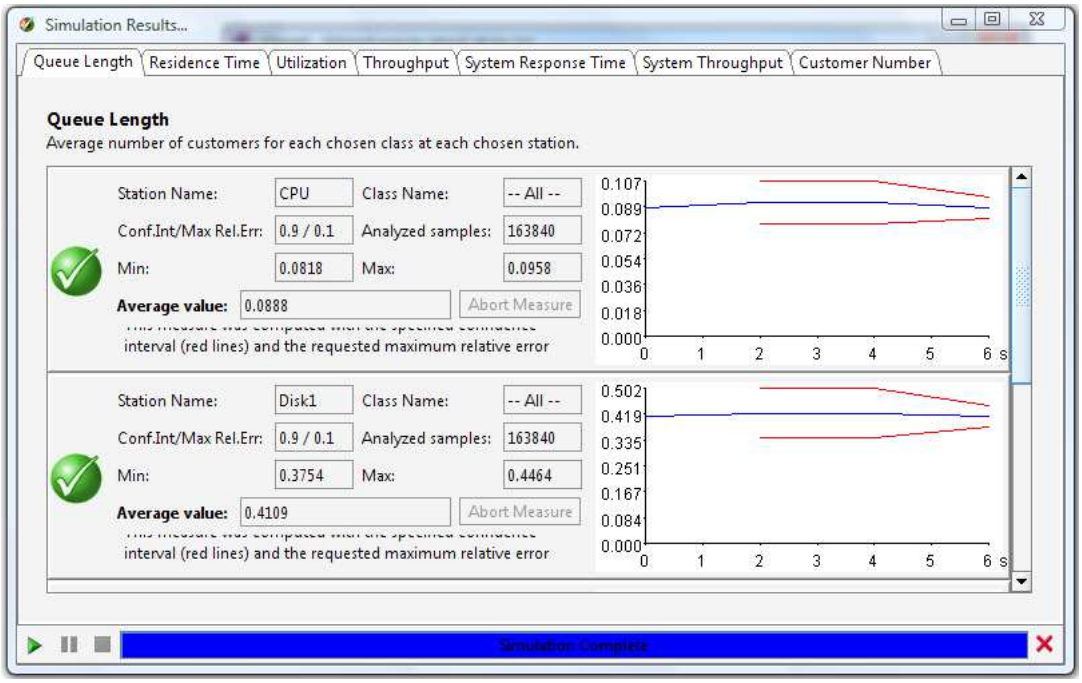


Figure 3.121: Progress of the simulation (average values and confidence intervals)

Before to start the simulation we have to define the performance indices. Our goal is to measure the equilibrium state of the population mix, thus we use the *System Number of Customers per class.*
Click on and set the performance indices as shown in Figure 3.137.

Now click on to start the simulation. As shown in Figure 3.138, the equilibrium state of the population mix is about $[0.666, 0.333]$.
For this simple model the result obtained with simulation can be validated in an analytical way.
Indeed, considering the matrix C defined by Equation 3.10 and indicating as β_1 the percentage of jobs of *Class0* and β_2 the percentage of *Class1* at the equilibrium, the following equations hold

$$\begin{cases} \beta_1 = 0.6\beta_1 + 0.8\beta_2 \\ \beta_2 = 0.4\beta_1 + 0.2\beta_2 \\ \beta_1 + \beta_2 = 1 \end{cases} \tag{3.11}$$

The solution of the system of equations (3.11) is β_1=2/3 and β_2=1/3, thus the *population mix* vector is {2/3 , 1/3}. We found these values also from simulation.

3.17 jSIM Log

The jSIM Log is a debug log file for the JSIM*graph* to understand the anomalies in the tool. If the tool crashes during simulation or if the user finds some inconsistency in the results of the simulation then they can upload this debug log file, so that the developers can understand and fix the problem. The logging of the events are done during the simulation. The file is written to append to a maximum size of 512KB, then the older lines are discarded in order to maintain the maximum size.

3.18 Command line for JSIM*graph*

It is possible to start a simulation JSIMgraph from *command line*. The only file needed is the *jsim/jsimg/xml* file. The simulation engine can be started with the *JMT.jar* file invoked as following:
`java -cp JMT.JAR jmt.commandline.Jmt sim <File.xml> [options]`.
The options are:
`-seed 1234` : sets the simulation seed to 1234
`-maxtime 60` : sets the maximum simulation time to 60 seconds .

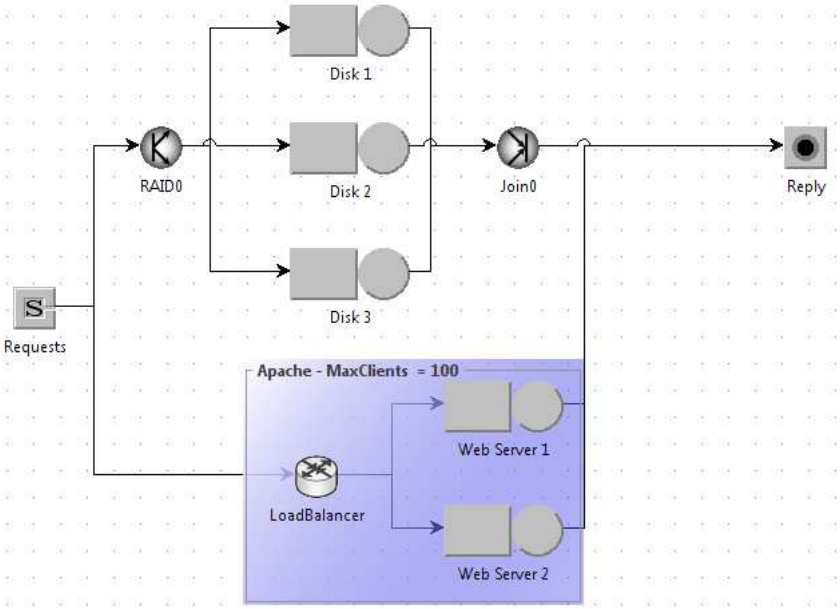


Figure 3.122: Example 2 - A system with a multiclass workload, a RAID0 storage server (modelled with `Fork` and `Join` Stations), and a web server consisting of two systems used in parallel, with a limited number of requests in execution (modelled with a Finite Capacity Region)

Example of a command line:
`java -cp JMT.JAR jmt.commandline.Jmt sim c:\mymodelname.jsimg -seed 4546 -maxtime 120`
The *JMT.jar* file is located in the same directory where the JMT tool is installed.
As for the MVA Solver, the `File.xml` parameter is a well formed XML File according to the *JMTmodel.xsd* schema described in subsection 2.6.1. At the end of the computation, performance indices will be placed into the **solutions** element of a new file which has the same filename with the input file but with "-result.jsim" appended.

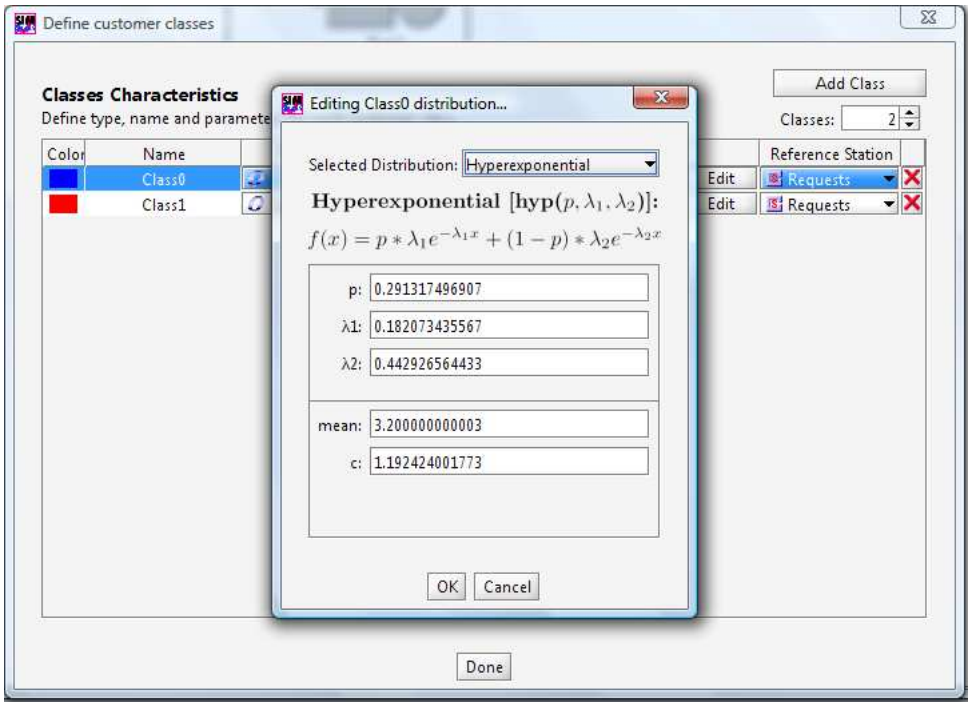


Figure 3.123: Definition of the hyperexponential distribution of *Class0* customers

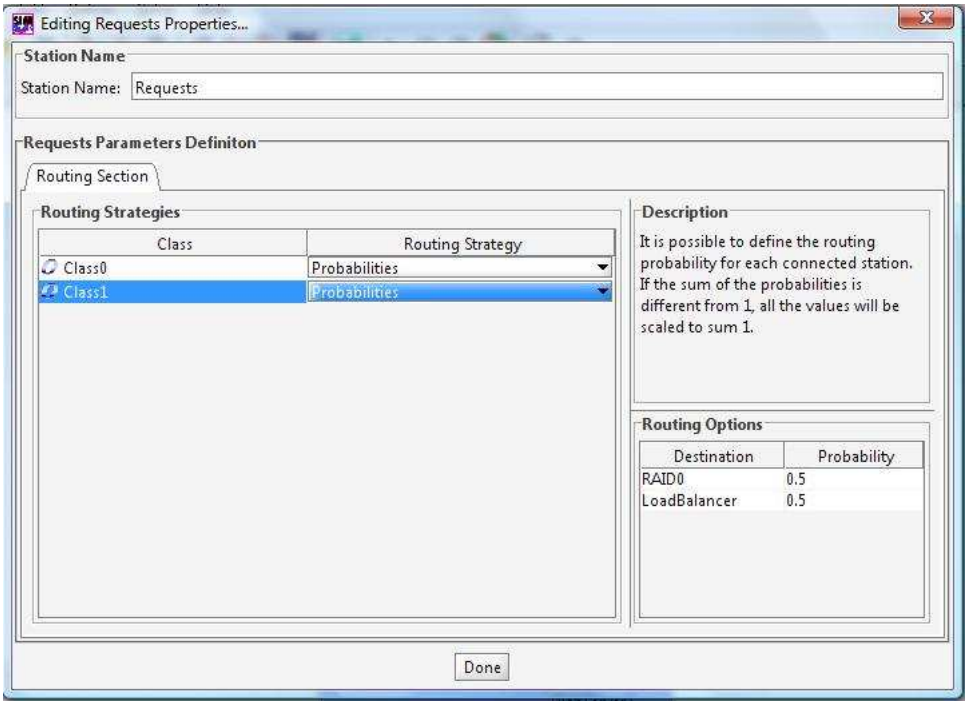


Figure 3.124: Setting the `Routing strategy` (*probabilities*) in output of station *Requests* for Class1 customers

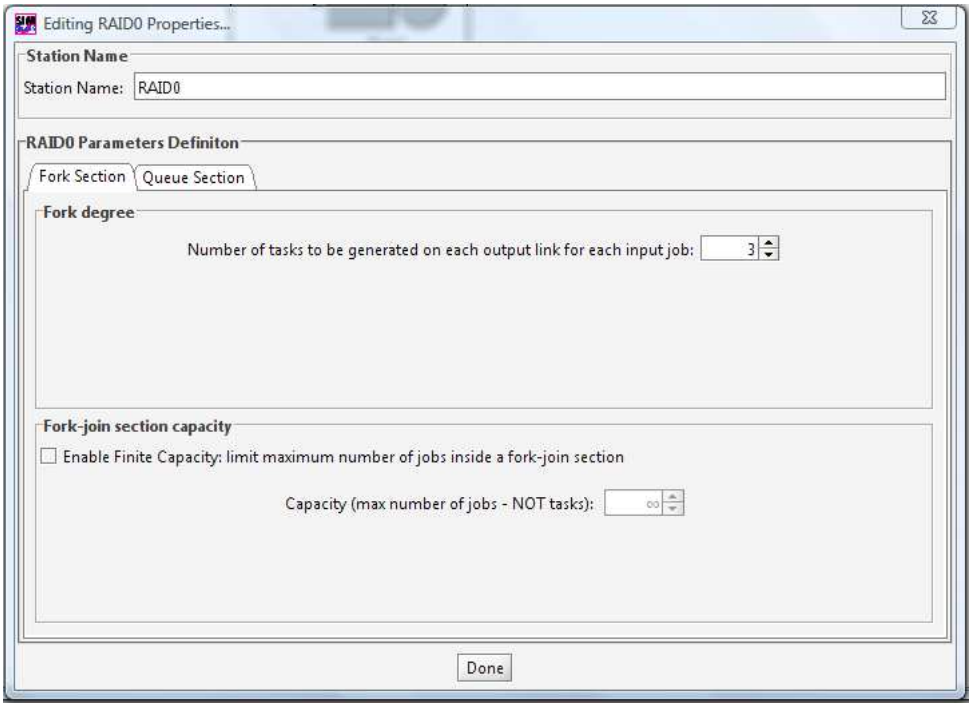


Figure 3.125: Setting the RAID0 *Fork* station properties

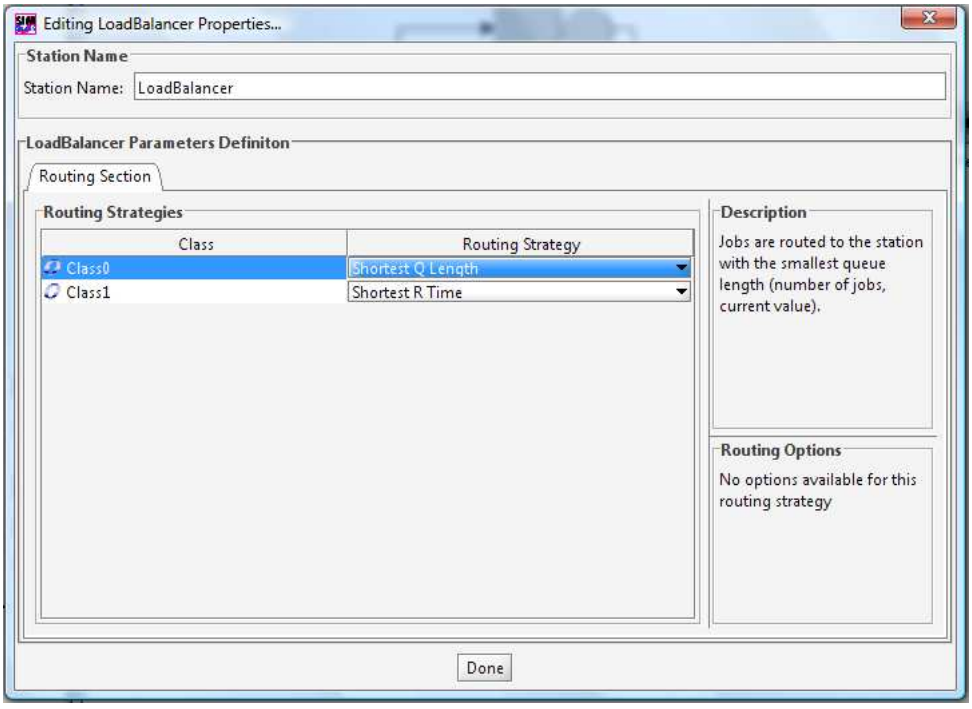


Figure 3.126: Edit `Loadbalancer` properties

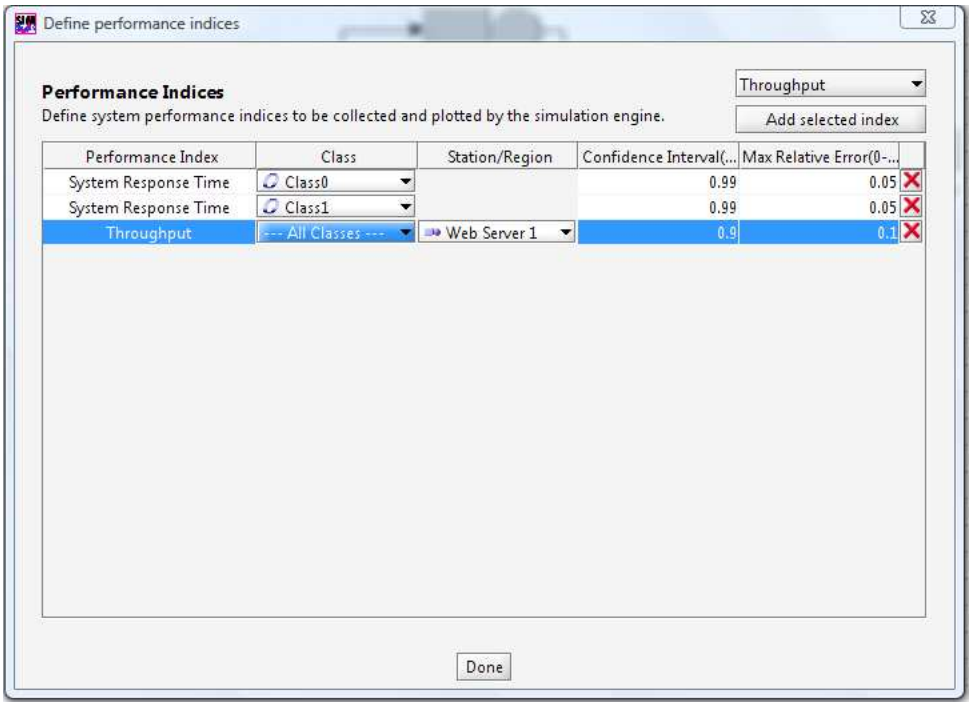


Figure 3.127: Definition of the performance indices, Confidence Interval, and Max Relative Error.

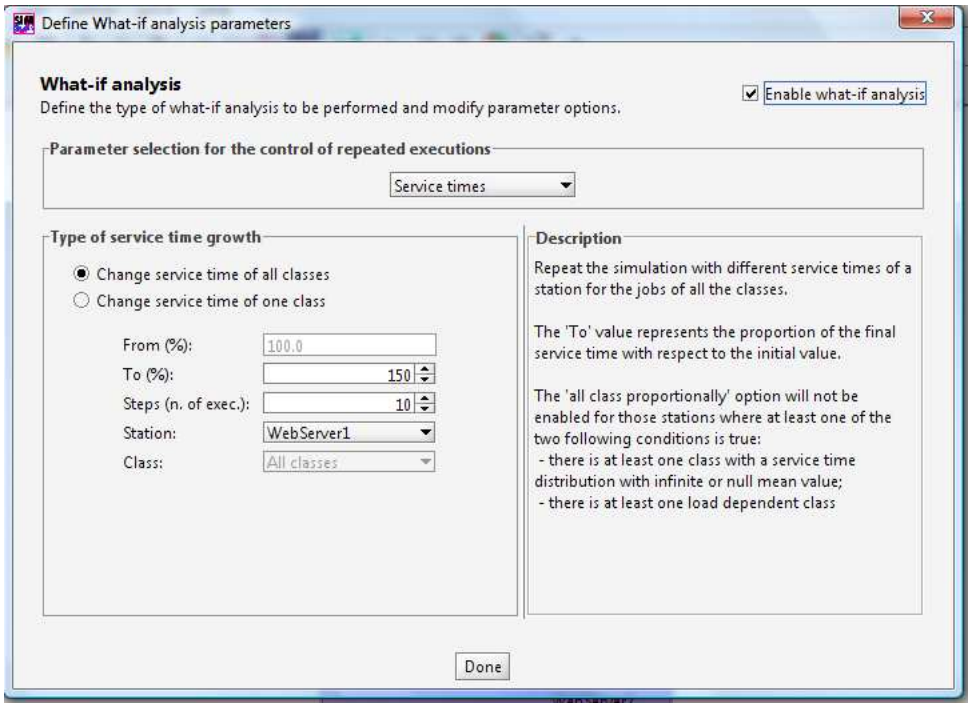


Figure 3.128: Setting the `What-If-Analysis` parameters.

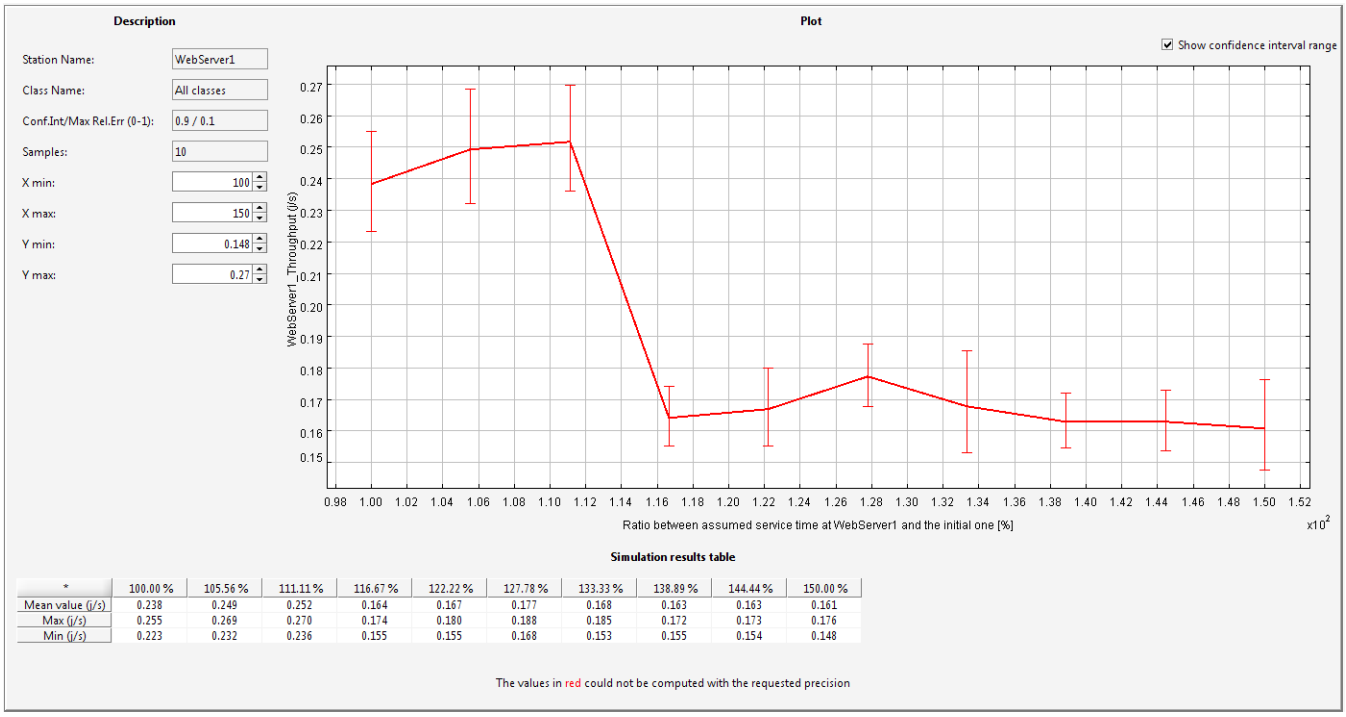


Figure 3.129: Throughput of the station *WebServer1* for all the classes of customers.

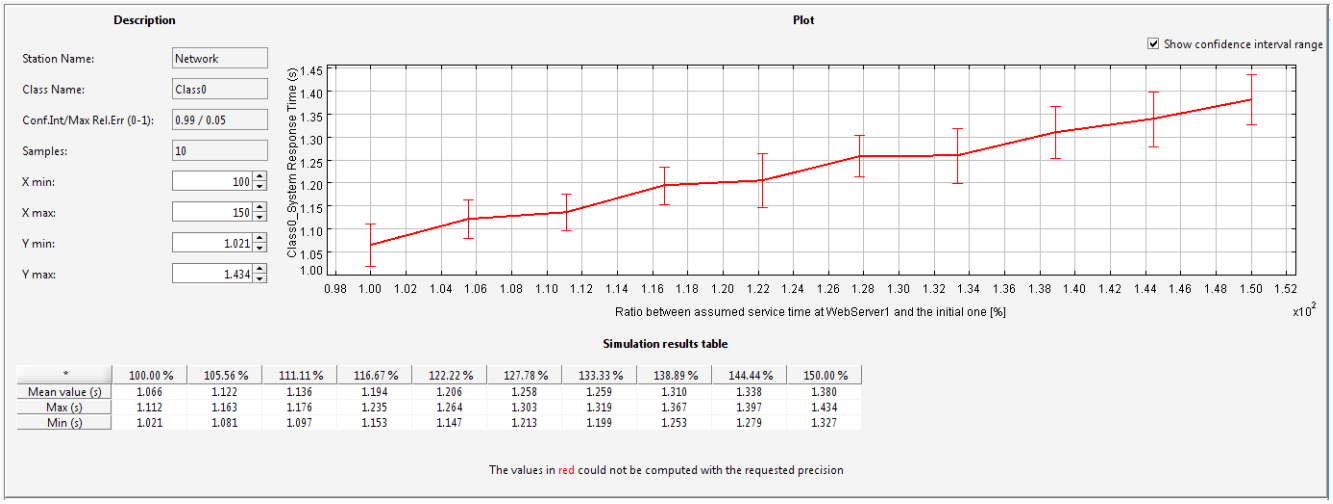


Figure 3.130: Behavior of System Response Time for *Class0* customers.

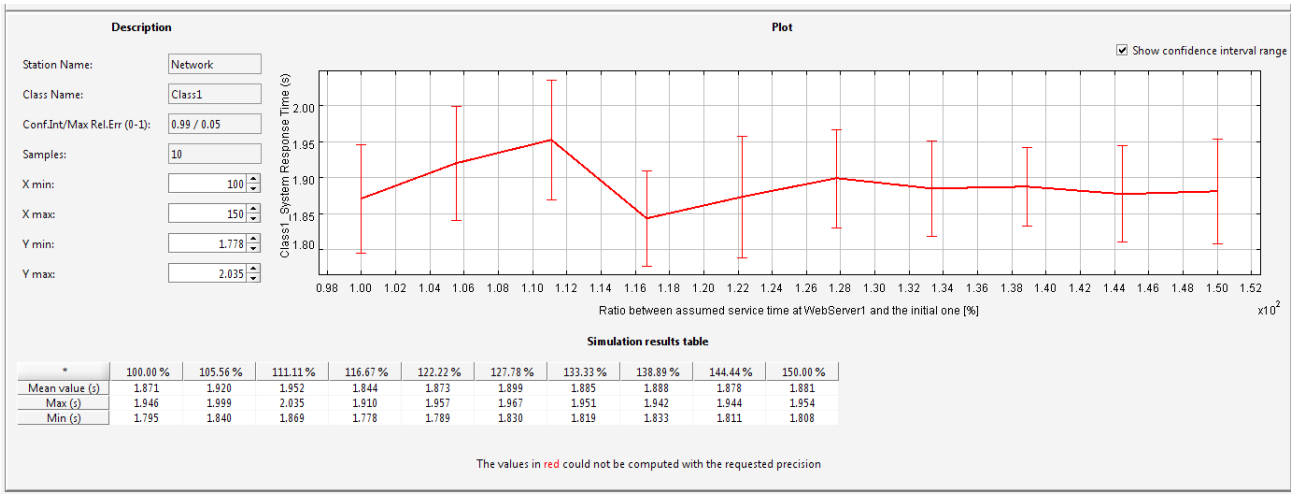


Figure 3.131: Behavior of System Response Time for *Class1* customers.

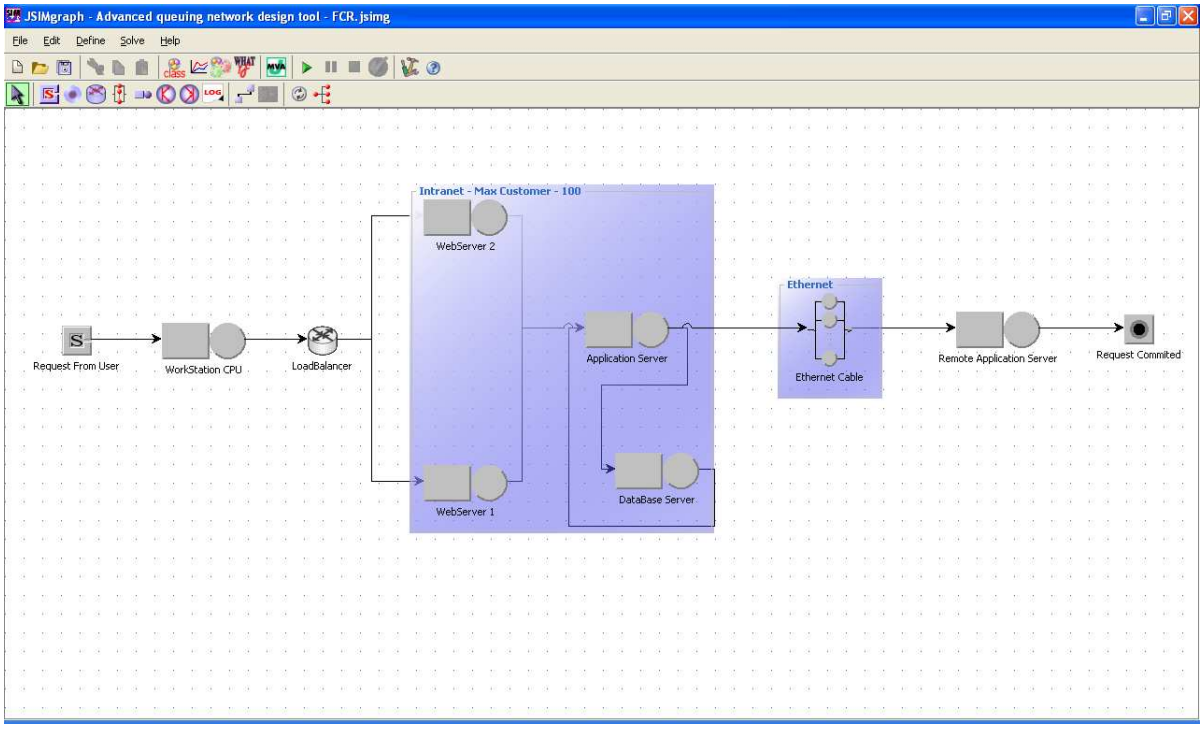


Figure 3.132: Example 3 - Model with 2 different FCRs. The first one, labelled as `Intranet`, allows maximum 100 customers to be concurrently in execution, while the second one, labelled as `Ethernet`, allows only one customer maximum in execution.

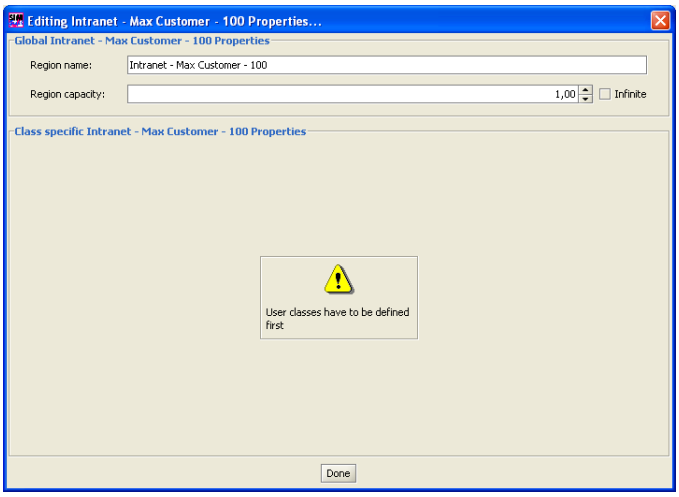


Figure 3.133: Example 3 - Setting capacity in FCR.

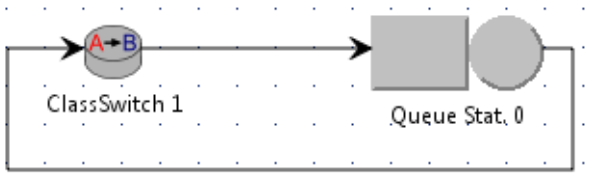


Figure 3.134: Example 4 - Two-class closed model with a class-switch station.

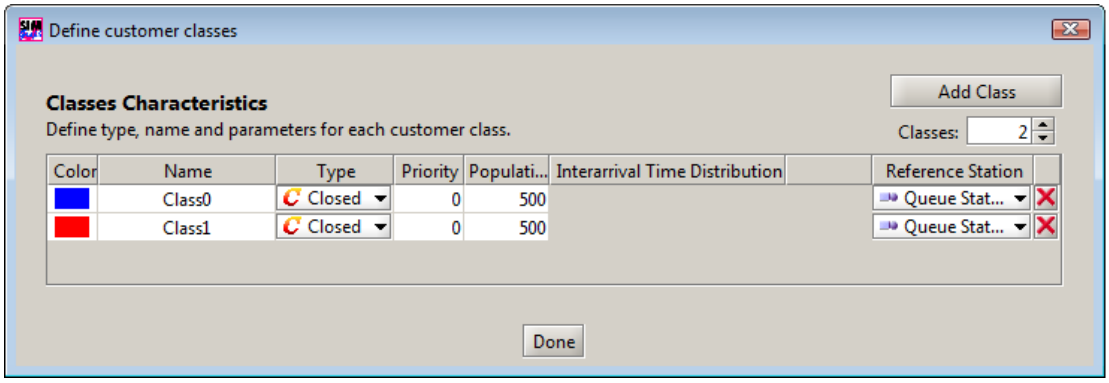


Figure 3.135: Example 4 - Initial state definition.

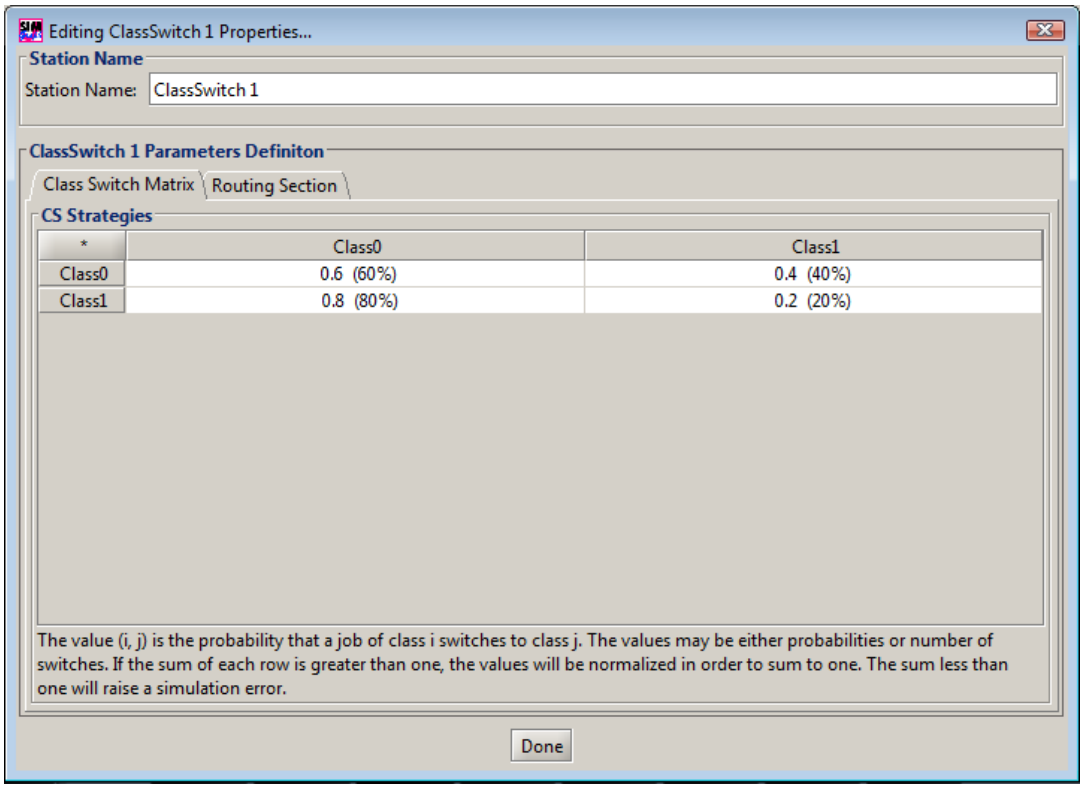


Figure 3.136: Example 4 - Class switch matrix.

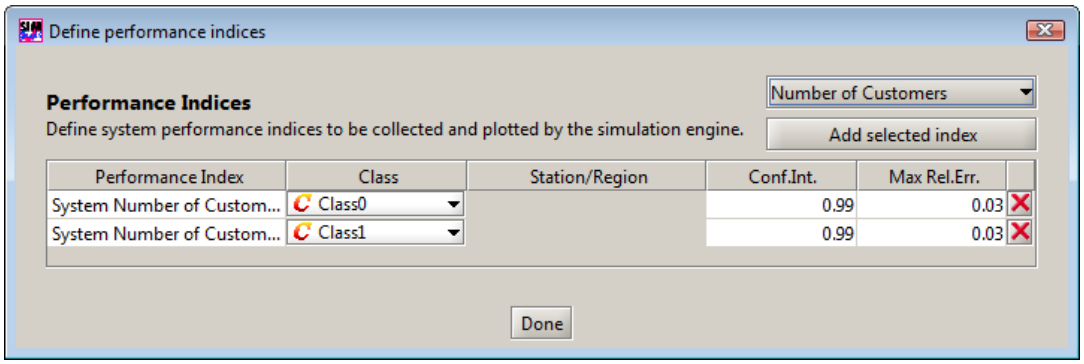


Figure 3.137: Example 4 - Performance indices definition.

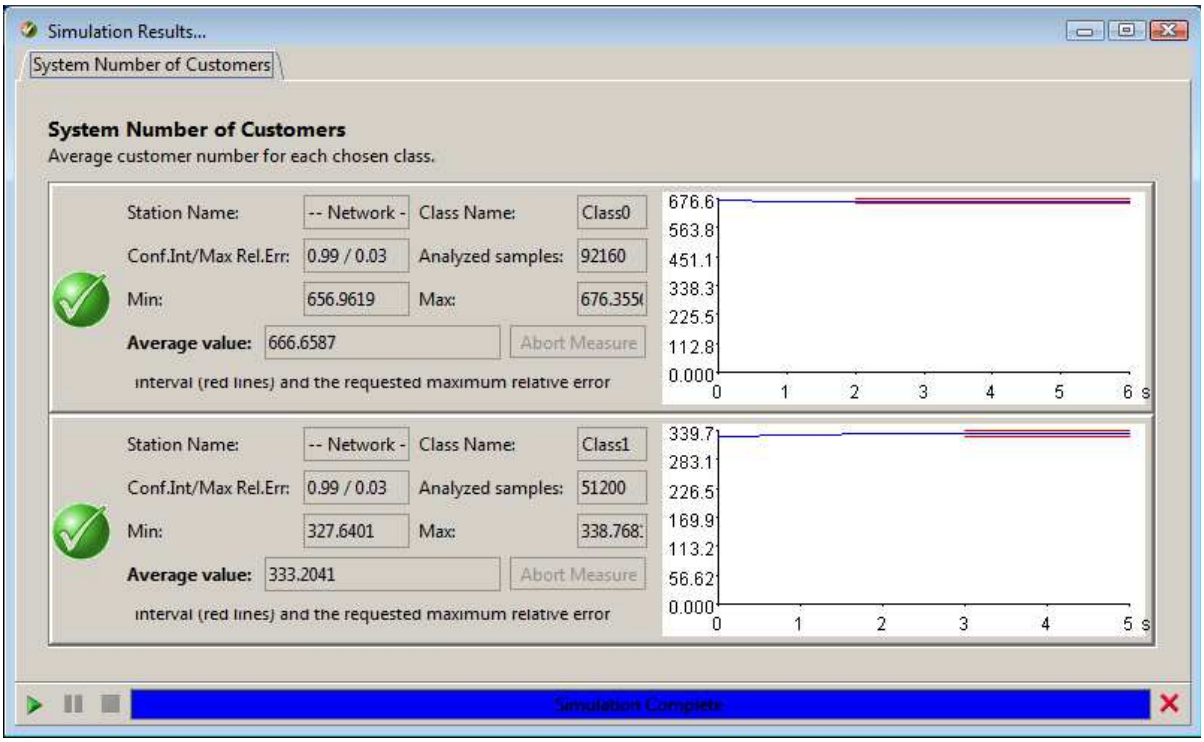


Figure 3.138: Example 4 - Results.

3.19 The XML format of JSIM models

The XML file tree structure generated by the JSIMg is shown in Fig.3.139.
A description of the tags and attributes is given in Tables 3.6 and 3.7.

Table 3.6: Tags and Attributes of the XML file of JSIMg (1)

Tag	Attribute	Value	Description	Child Tag
Archive	name	string	jsimg file name	sim, jmodel,
	timestamp	time	jsimg generation time	
sim	disableStatisticStop	true/false	whether set conditions to stop simulation	results
	logDecimalSeparator	,/.	set ,/. for decimal numbers	
	logDelimiter	,/;/space/tab	set delimiter between values generated from result csv file	userClass, node, measure, connection, preload.
	logPath	string	JMT path for saving files	
	logReplaceMode	double	no log data replacement	
	maxSamples	double	define max number of samples in simulation	
	maxSimulated	double	number of sample control condition when disableStatisticStop is true	
	maxTime	double	duration control condition when disableStatisticStop is true	
	name	string	jsimg file name	
	polling	double	Animation update interval in second	
	seed	double	Simulation random seed	
userClass	customers	double	population of the class	NA
	name	string	class name	
	priority	double	whether has priority to enter queues	
	referenceSource	string	starting or reference station	
	type	open/closed	define user type	
parameter	array	true/false	define whether parameters are in an array like user class.	value, referClass
	classPath	string	java class path of the software	
	name	string	parameter name	
subParameter	classPath	string	java class path of the software	value, referClass
	name	string	sub-parameter name	
value	NA	double	a value for parameter/subParameter	NA
referClass	NA	string	a value for referring class	NA

Table 3.7: Tags and Attributes of the XML file of JSIM*graph* (2)

measure	alpha	0.01-0.99	a percentile for simulation progress	sample (only for result)
	name	string	performance indice measure name	
	nodeType	string	type of node of its measure	
	precision	double	parameter to control measure precision	
	referenceNode	string	refer node name	
	referenceUserClass	string	refer class name	
	type	string	define measure type; Can be number of customers, queue time,residence time, response time, utilisation, throughput, drop rate, system throughput, system response time, system drop rate, system number of customers, system power, throughput per sink, response time per sink.	
	verbose	true/false	check whether this measure result is verbose	
	analyzedSamples	double	record number of analysed samples	
	discardedSamples	double	record number of discarded samples	
	finalValue	double	value of this measurement indice	
	state	double	define state of simulation	
connection	source	string	source node of the connection	NA
	target	string	target node of the connection	
preload	NA	double	set preload station and population of the model	station-Population, class-Population
stationPopulations	stationName	string	preload station name	NA
classPopulation	population	double	define number of class customers	NA
	refClass	string	refer to a class name	
jmodel	NA	NA	a super tag for creating model layout	station, userClass
station	name	string	node/station name	position
position	rotate	true/false	whether the station can rotate or not	NA
	x	double	x axis coordinates	
	y	double	y axis coordinates	
results	logDecimalSeparator	,/.	separator of decimal number	measure
	logDelimiter	,/./space/tab	separator of results in csv file	
	pollingInterval	double	animation interval time in second	
sample	lastIntervalAvgValue	double	record last interval avg value in the simulation	NA
	meanValue	double	mean value of the measure for this sample set	
	simulationTime	double	record time spent in simulation	

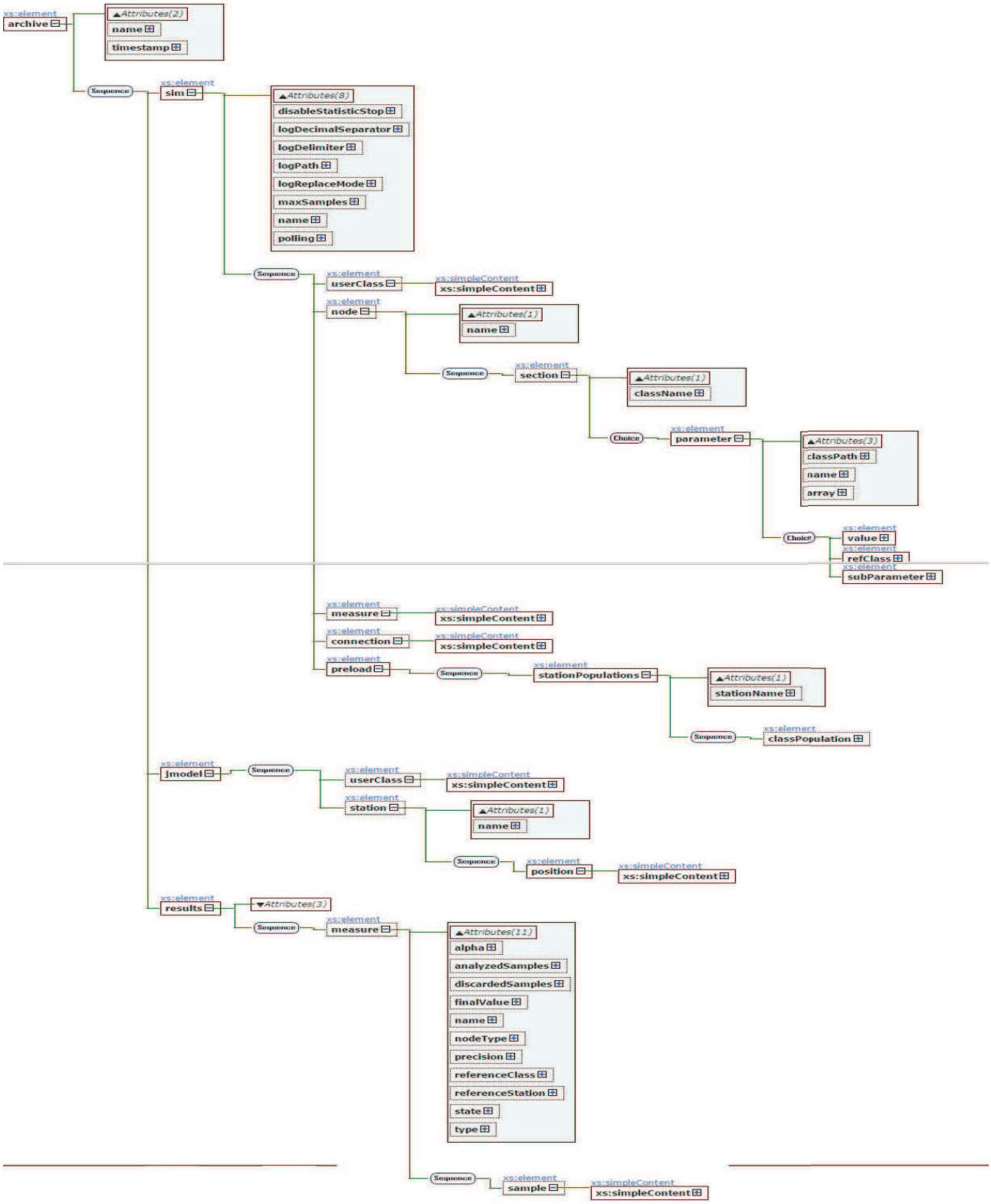


Figure 3.139: XML tree structure of a JSIMg file

3.20 The JSIMgraph Templates

3.20.1 What are the *Templates*

Templates are pre-designed models, or sub-models, which are made available to JSIM*graph* users to simplify and speed-up their modeling efforts. The typical use of a template is to automated the long sequence of operations required to create a complex simulation model. For example, creating a Fork/Join subsystem with a large number of queues may require a lot of time. Using the `Parallel model` template, the user can instead generate the Fork-Join subsystem in a few clicks.

JMT currently supports two different types of templates:
1) *official templates*, offered by the JMT team through the official `sourceforge` server.
2) *community templates*, offered by third parties who want to enrich the functionalities of JMT.
The *official templates*, will be available to JMT users that can download them and will be automatically installed in their systems in the folder `Users/<username>/JMT/templates/` created by JMT. Section 3.20.2 describes the steps to download these templates.
The operations required to write a new template and to download the *community templates* located on users' servers will be described in a separate document.

3.20.2 Download of *Official* Templates from JMT `sourceforge` site

The main steps for the download of templates from the official JMT `sourceforge` site is shown in Fig.3.140.

Open the main window of JSIMgraph. Press button T and the window of Fig.3.141 will appear. To see all

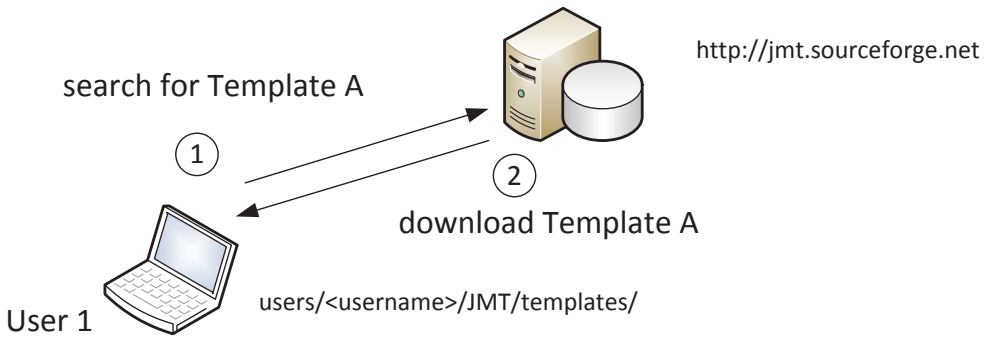


Figure 3.140: Main steps for the download of *Official Templates* from the JMT `sourceforge` site

the Templates available press `Add/see all Templates` button and window of Fig.3.142 with the list of all the available templates, grouped in categories, will appear.
Select the Template that you want to download (e.g., Parallel Model in Fig.3.142) and press the red button `Download`. From this list you may see that the `RAID` templates have already been downloaded. All the templates that have already been downloaded appear in the window of Fig.3.143 Select the template you want to instantiate, e.g., the `Parallel Model` template, and press the `Instantiate` button. The window of Fig.3.144 will appear that allows the parameterization of the template. Depending on the template structure, some parameters can be set directly from this window (e.g., the number four of the parallel servers), while others can be set in the windows that can be opened through the buttons shown in the bottom. Once all the parameters are set, it is sufficient to press `Insert` and an instance of the template will be incorporated in the main window of JSIMgraph (Fig.3.145). Clearly, this instance may be a complete model itself or a part of a bigger model that a user may integrate with other components or other templates.

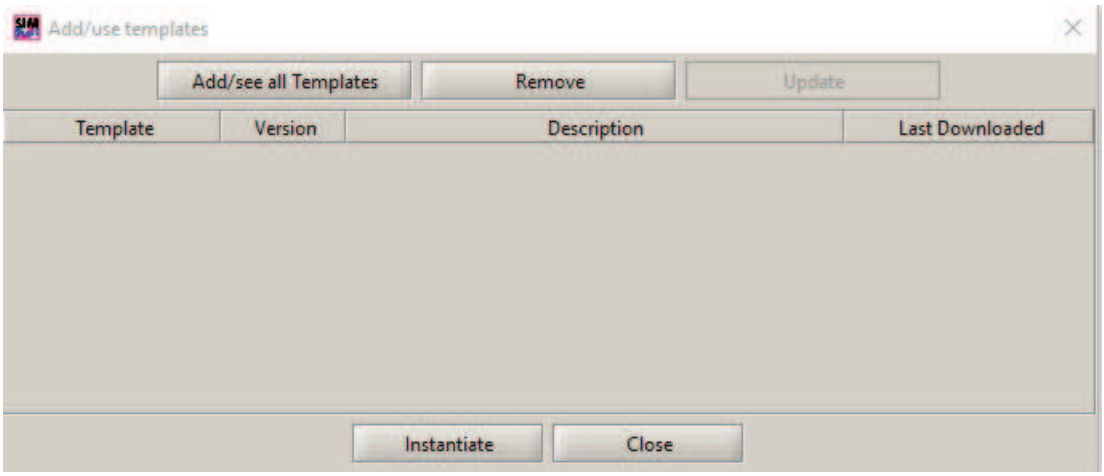


Figure 3.141: Initial window for the management of Templates.

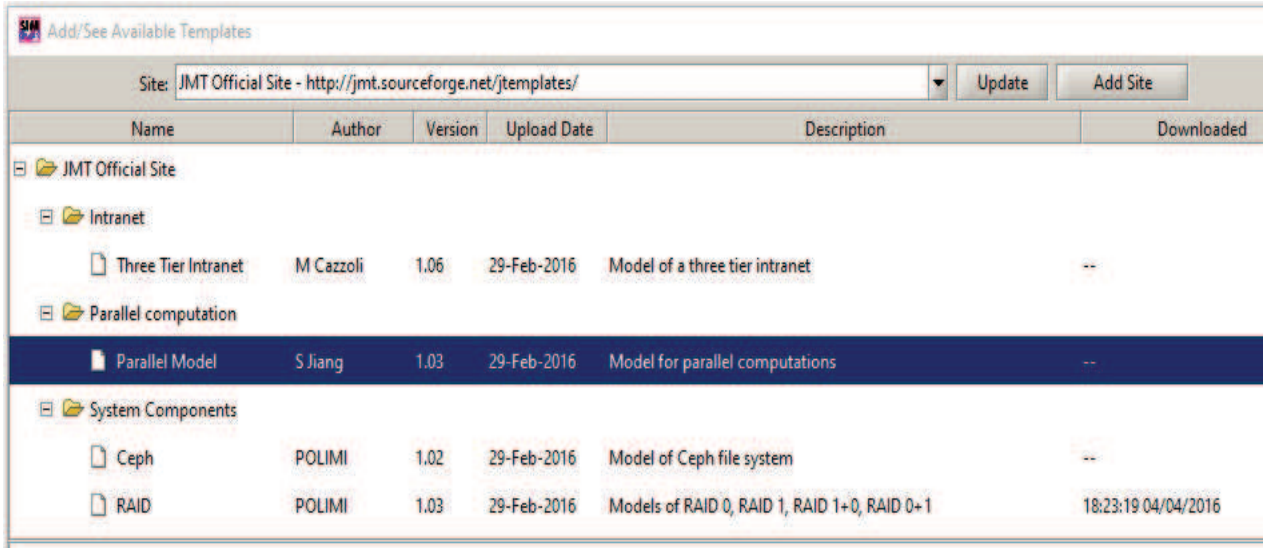


Figure 3.142: List of the templates available on the sourceforge JMT Official Site.

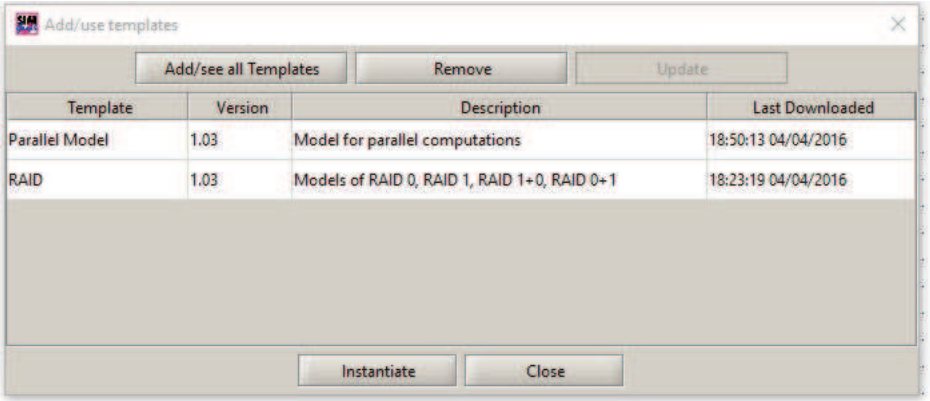


Figure 3.143: Templates already downloaded and available in the folder `users/<username>/JMT/templates` of the user's system.

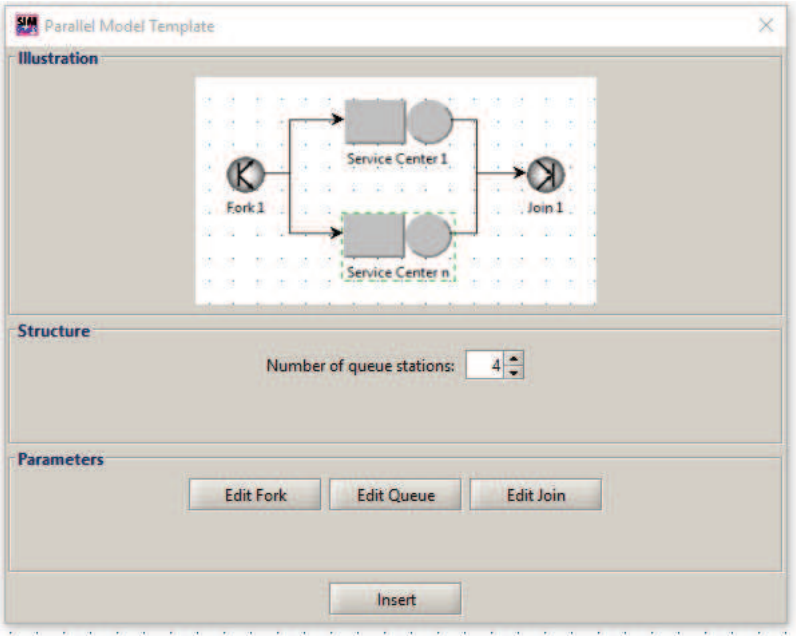


Figure 3.144: Window for the parameterization of the `Parallel Model Template`. Four parallel servers have been requested.

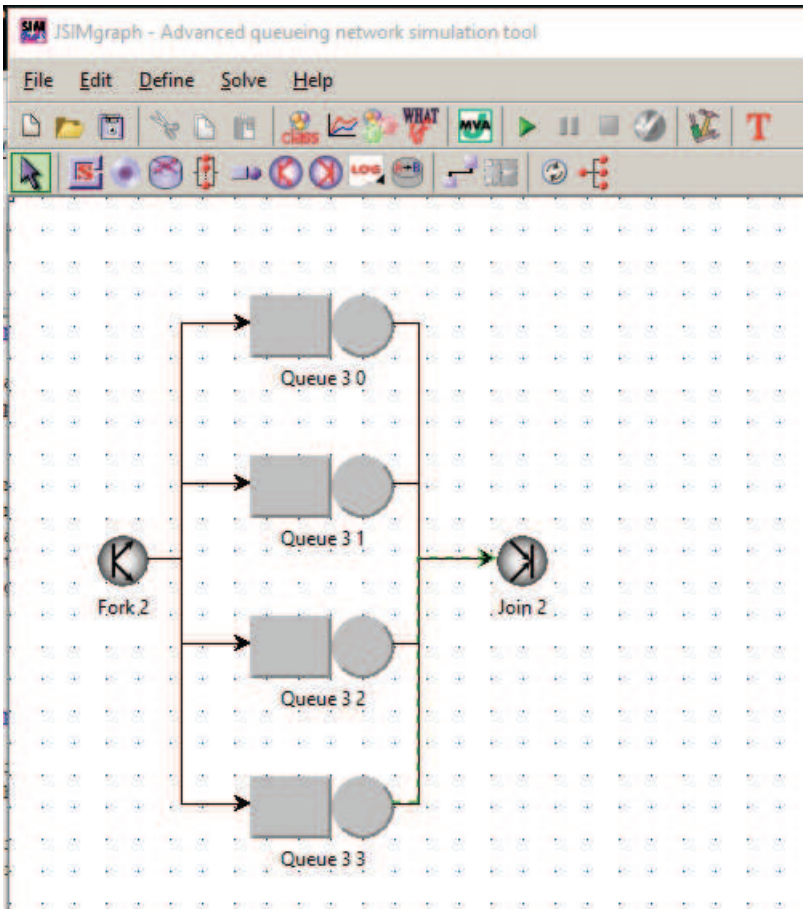


Figure 3.145: Instance of the `Parallel Model Template` with the four parallel servers requested.

Chapter 4

JSIM*wiz* (Simulation - Textual interface)

4.1 Overview

Simulation can be applied when the characteristics of the network to be analyzed make analytical techniques no longer useful for its performance evaluation. A simulator is a dynamic model able to follow the evolution in time of both the system complexity it represents and the pattern of arrival requests to be executed.
In the JMT *suite* a discrete-event simulator JSIM for the analysis of queueing network models is provided. It can be used through two interfaces: alphanumerical (JSIM*wiz*) and graphical (JSIM*graph*). **The manual of JSIM*graph* is much more detailed than this one. If some information are omitted or unclear in this chapter you may consult the chapter of JSIM*graph*.**

An intuitive user interface allows even an unexperienced user to build with JSIM*wiz* system models with sophisticated characteristics. It helps the users to perform an evaluation study in two ways. Firstly, critical statistical decisions, such as transient detection and removal, variance estimation, and simulation length control, have been *completely automated*, thus freeing the users from taking decisions about parameters s/he may not be familiar with. The simulation is automatically stopped when all performance indexes can be estimated with the required accuracy. Secondly, a user-friendly graphical interface allows the user to describe, both the network layout and the input parameters. Furthermore, the graphical interface also provides support for the use of advanced features (several of them are for networks with very general characteristics, usually referred to as *non-product-form* networks) like fork and join of customers, blocking mechanisms, regions with capacity constraints on population, state-dependent routing strategies, user-defined general distributions, import and reuse of log data. A module for *What-If Analysis*, where a sequence of simulations is run for different values of control parameters, particularly useful in capacity planning, tuning and optimization studies, is also provided.

The simulation engine performs on-line the statistical analysis of measured performance indices, plots the collected values, discards the initial transient periods and computes the confidence intervals.
All the plots generated by a simulation can be exported in vector (e.g., eps, pdf) or raster (e.g., jpg, png) image formats.

Main Features

Arrival rates for open classes of customers generated by Source stations and station **service times** (for any type of station in open and closed models) can be generated according to the following distributions: Burst (General), Burst (MAP: Markovian Arrival Process), Burst (MMPP2: Markov-Modulated Poisson Process of order 2), Coxian, Deterministic, Erlang, Exponential, Gamma, Hyperexponential, Lognormal, Normal, Pareto, Phase-Type, Replayer, Uniform, Weibull

The following **Queueing disciplines** are available: First Come First Served (FCFS), Last Come First Served (LCFS), all with or without priority.

Routing of the customers in the network, i.e., the path followed by the requests among the resources, can be described either probabilistically or according to the following strategies: Fastest Service, Least Utilization, Load Dependent Routing, Random, Round Robin, Join the Shortest Queue, Shortest Response Time (the values of the control parameters are evaluated on the stations connected in output to the considered one). These strategies can be further combined among themselves through the use of a *routing station*.

Other peculiar features of the simulator are:

- Load dependent service time strategies
- Fork-and-join stations to model parallelism
- Simulation of complex traffic pattern and service times (e.g., burst)
- Blocking regions (in which the number of customer is limited)
- What-if analysis (with various control parameters)
- Customization of default values
- Import/Export of the model from/to JMVA, the exact solver (when the required analytic assumptions are satisfied)
- Logging of the job flows in any part of the model (*Logger* station)
- Graphical visualization of the evaluated performance indices together with their confidence intervals
- Automatic transient detection and removal
- Automatic stop of the simulation when all performance metrics can be estimated with the required accuracy.

JSIM*wiz* has a modular Java-based architecture that allows the introduction of new Java classes in the simulation engine without any modification to the source codes of the other classes.

4.1.1 Starting the discrete-event simulator

Selecting  button on the starting screen, Figure 4.1 window shows up. Three main areas are shown:

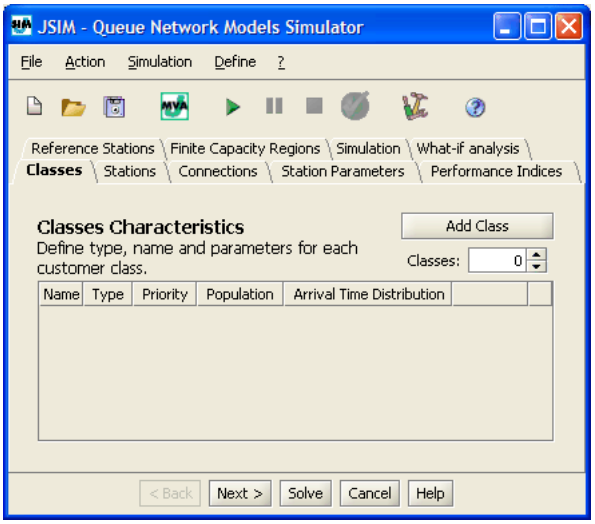


Figure 4.1: The home window of JSIM*wiz* simulator

Menu : it is organized into three groups of functions. To use a menu, click on the menu heading and choose the appropriate option. For the description of menu entries, see section 2.4

Toolbar : contains some buttons to speed up access to JSIM functions (e.g. New model, Open, Save...). If you move the mouse pointer over a button a tooltip will be shown up.

Page Area : this is the core of the window. All JSIM parameters are grouped in different tabs. You can modify only a subset of them by selecting the right tab, as will be shown later.

4.2 Defining a new model

To define a new model, select the New command from the File menu, or the button  or use the shortcut CTRL+N. A new model is automatically created every time JSIM is started.

The following parameters must be defined in order to complete the new model:

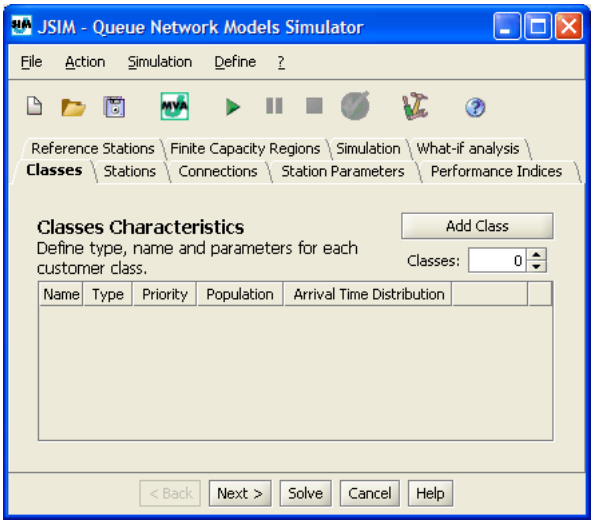
- Classes
- Stations
- Connections

- Station parameters *
- Performance indices
- Reference stations
- Finite capacity regions *
- Simulation *

To set each parameter, follow the user manual step by step.
NOTE: if no values are provided for the parameters marked with *, default values will be used and the simulation will run anyway.

4.2.1 Define Classes

Customer classes identify different customer behavior and characteristics, such as the type (closed or open), the size of the customer population (for closed classes) or the interarrival time distribution (for open classes). They can be set using the `Classes` tab during the creation of a new model.

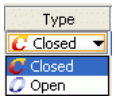


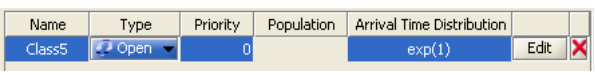
Adding a Class

Classes must be explicitly added to the model, either one at a time by clicking the Add Class button, or by selecting directly the final number of classes desired in the form Classes: 0. The newly added classes will be listed with deafult parameters.
Double click on the default name (ClassN) to change it.
Each new class has a priority in the system. A smaller number indicates a lower priority. Default value is 0 and it can be changed by double clicking on the corresponding area.

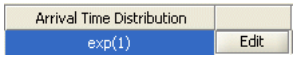
Defining the Class Type: Open Classes

After adding a class and possibly changing its name and priority, you must choose the type of customers comprising the class. Classes are created Closed by default, so if you want an Open class, select the type `Open` in the menu. The class characteristics looks like this now

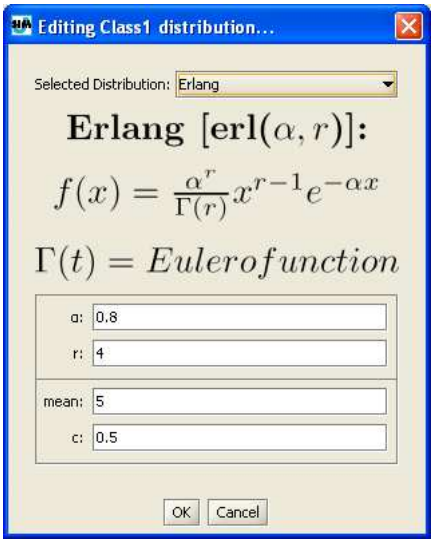




Open classes describe customer populations that vary during time, therefore they are best characterized by the probability distribution of the interarrival time, rather than by a constant number of customers. The default Interarrival Time Distribution is exp(1) (Exponential Distribution with λ =1).
To change the Interarrival Time Distribution click the `Edit` button



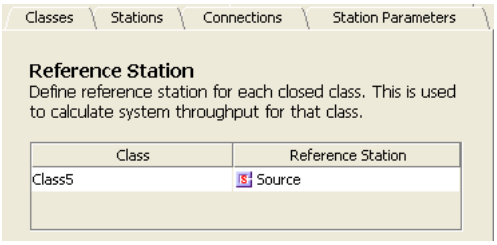
The following window will appear



Click on the drop-down menu `Selected Distribution` to choose one of the following distributions (see section 3.6):

- Burst (General)
- Burst (MAP)
- Burst (MMPP2)
- Coxian
- Deterministic
- Erlang
- Exponential
- Gamma
- Hyperexponential
- Normal
- Pareto
- Phase-Type
- Replayer
- Uniform

In each case, it is possible to configure the distribution parameters as you wish, or use the default values. Parameters that are related each other are automatically adjusted if one is modified (for example, the figure shows how for an Erlang distribution, if you set the (α, r) pair, the (mean, c) pair is automatically adjusted to the correct value). The `Replayer` distribution allows you to provide data traces from files. Click OK when you are done to return to Class parameters definition.
The final step in defining a class is the definition of the Reference Station, i.e., that station in the model with respect to which the performance indices will be computed. For Open classes there is no choice since the Source station is the unchangeable default Reference Station used to compute System Throughput for all the Open classes.

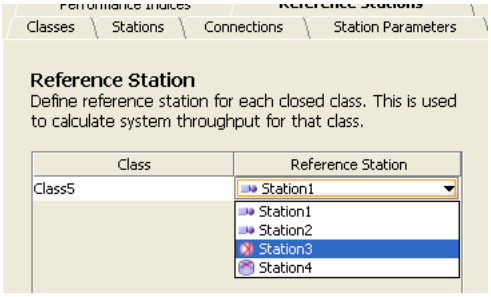


Defining the Class Type: Closed Classes



Classes are created `Closed` by default, so there is not need to change the Type. Priority can be changed as in the Open class case. The population size is the parameter that characterizes a Closed class. It is fixed and does not change for the entire life of the system. By default is 1 and it can be changed by clicking on the corresponding area in the class properties matrix.

The final step is to define a `Reference Station` for the class that will be used to compute the performance indices selected for the class. Use the `Reference Station` tab menu to select the `Station`. All stations but the `sink` can be used as `reference station` for a closed class.

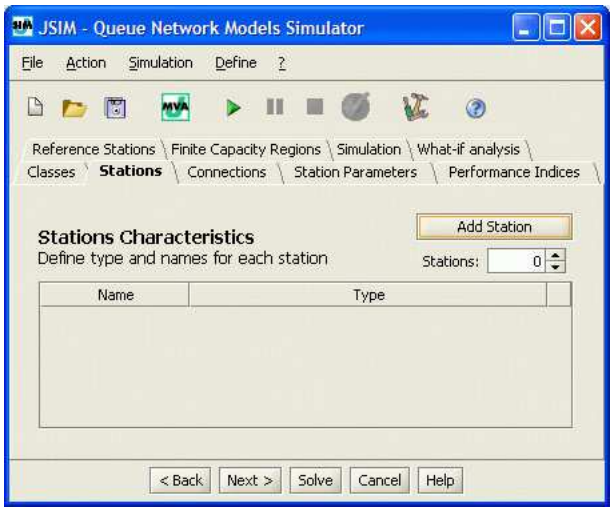


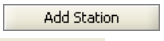
Distributions

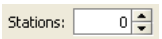
See section 3.6

4.2.2 Define Stations

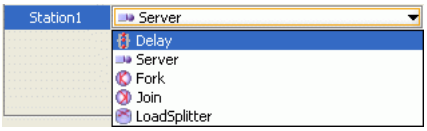
When you create a new model, you must define the Stations from the `Stations` page of the tabs menu.



You can insert one station at a time, by pressing the 

button, or multiple stations all at once, by choosing the desired number in the input form 

The added stations will be listed in the page. Various types of stations are available, the default type being *server*. If you want to remove any of the stations, select the target and press the corresponding delete button. Stations are name Station1, Station2, and so on by default. If you want to assign model-relevant names, left-clik on a name to change and replace it. Similarly, if the default station type Server is not what you need, select from the drop-down menu of each station the correct type.



You can choose among the following types of station:

1. **Delay station:**
Customers that arrive at this station are delayed for the amount of time that defines the station service time. They do not experience any queueing, since a delay station is modelled as a station with an infinite number of servers with identical service time. Because customers never have to wait for service, response time at delay stations is equal to service time, $R_i = S_i$. Furthermore, the queue length corresponds in this case to the number of customers receiving service since there is no waiting in queue: $N_i = R_i\ X_i = S_i\ X_i = U_i$. The utilization of the delay station represents the average number of customers receiving service (i.e., in think state) so it may be *greater* than 1! Delay stations are used when it is necessary to reproduce some known average delay (with the selected distribution). Application of delay stations may be to model transmission time (download, upload, ...) of large amounts of data over a network (Internet, lans, ...) or to model users think time at the browsers.
In the `Station Parameters` tab menu page, you can modify:

   Service Section: in this section you can choose the service time distribution for the station. For more information, see Service section section 4.2.2.

   Routing Section: in this section you can specify how serviced customers should be routed to the next station they will visit in the model. For more information, see Routing section section 4.2.2.

2. **Server station:**
The Server station is one of the most important components in a queuing network model. Service stations represent the service facilities of the system being modeled. A server station provides the required service to its customers. Server stations may have any (finite) number of servers. If all the servers in the service station are busy when a customer arrives at the station, the arriving customer is put in a waiting queue until its turn to receive service from the first available server is up. The queueing discipline decides which customer is served next when a server becomes free. Therefore, the response time at server stations includes service time and queuing time. Server stations may have more than one server. Their number is a parameter to be specified (default is 1).

   In the Station Parameters tab menu page, you can modify:

   Queue Section: in this section you can choose the type of queue (finite or infinite) and the policy used to select the next customer to be served. For more information, see Queue section section 4.2.2.

   Service Section: in this section you can choose the service time distribution for the station. For more information, see Service section section 4.2.2.

   Routing Section: in this section you can specify how serviced customers should be routed to the next station they will visit in the model. For more information, see Routing section section 4.2.2.

3. **Fork station:**
A JSIM Fork station is simply a station where jobs are split into tasks. No service is provided, therefore there is no service time specification. Tasks are then routed along the Fork station outgoing links. Unlike classical queueing theory fork-join queues, a JSIM Fork station is not associated with a join station automatically. Any combination of server stations, finite regions, fork-join, routing stations, etc., is possible after a Fork station. This feature allows the modeling of very general parallel behaviors, of which the traditional Fork-Join one is a special case. In JSIM the classical queueing theory Fork-Join queue behavior is obtained by connecting the Fork station to as many Server stations as the degree of parallelism requested, with one task per outgoing link. Each Server station is then connected to a Join station, where the job is recomposed. A Fork station is characterized by the forking degree, i.e., the number of tasks routed on each one of its outgoing links, and the capacity, i.e., the maximum number of jobs that can be served by the station simultaneously. Therefore, the number of sibling tasks a job is split into is given by the product of the number of outgoing links from the Fork station times the forking degree. Note that a finite station capacity makes sense only when there is a join station downstream from the fork station that can recompose the split jobs. Otherwise, inconsistencies in the model and subsequent simulation error, such as out-of-memory, may occur. No automatic checks are available at the moment that can identify such critical conditions. Both the forking degree and the capacity are section parameters to be specified in the

model. As an example, a Fork station with forking degree 1, connected to three Server stations, would split a job into 3 sibling tasks (this is a traditional Fork-Join like behavior). If no Join station is connected to any of the three Servers, a warning message is displayed since the model could quickly saturate due to the extra load (3 more jobs) created in addition to each job entering the Fork station.

In the Station Parameters page of the tabs menu,you can modify:

Fork Section: in this section you can choose the forking degree of a job on each outgoing link and the fork station capacity, i.e., the maximum number of jobs that can be served in parallel by the station. For more information, see Fork section section 4.2.2.

Queue Section: in this section you can choose the type of queue (finite or infinite) of the fork station in case its capacity is finite and jobs must wait before proceeding through the fork station. For more information, see Queue section section 4.2.2.

4. **Join station:** 
Join stations are complementary to fork stations. In classical queueing network theory, a task arrives at a join station from the corresponding Fork station. In JSIM, a Join station may have incoming links from stations other than the corresponding Fork one. A Join station has no service time, as it is only used to recombine the tasks a job had been previously split into and then route the job to some other station(s). When a task arrives at a join station, it waits until all its sibling tasks have arrived. At this time, the original job is recomposed and routed to the next station. If a job arrives from a station other than the corresponding Fork, i.e., a job that was not split, it is simply routed to the next station. In this case the Join station operates as a routing station.

   In the Station Parameters tab menu page, you can modify:

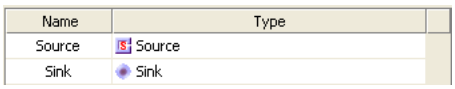
   Routing Section: in this section you can specify how serviced customers should be routed to the next station they will visit in the model. For more information, see Routing section section 4.2.2.

5. **Routing station:** 
A routing station is a dummy station, with service time equal 0, that is used to create more complex and sophisticated routing strategies by sending jobs through one or more such stations. For example, if we wanted two thirds of the incoming traffic at station Z to be randomly routed to either station A or B and the remaining third to go either to station C or D, depending on the shortest queue at the two stations, we could implement the following. Add a routing station, Y, among Z's output stations and define random routing for the three of them (A, B, Y). Then connect Y to C and D and define Join the Shortest Queue routing to C and D.

   In the Station Parameters tab menu page, you can modify:

   Routing Section: in this section you can specify how serviced customers should be routed to the next station they will visit in the model. For more information, see Routing section section 4.2.2.

6. **Source station:** 
If the model comprises at least an open class, a sink and a source stations are created by default, as they are necessary for the simulation and cannot be removed



Open classes are characterized by an infinite stream of jobs that can enter the system. Source stations are used to introduce jobs in the model. Their service time is the interarrival time of each customer class and as such, it is defined as a parameter of the class the job belongs to. The routing strategy defines the first station a newly created job will visit. Only open class jobs can be routed from source stations.

In the Station Parameters tab menu page, you can modify:

Routing Section: in this section you can specify how newly created customers should be routed to the next station they will visit in the model. For more information, see Routing section section 4.2.2.

7. **Sink station:** 
Open class customers leave the system once they have received all the service the need. Sink stations are

used to model customers leaving the system, as they enter the sink station but do not ever leave it. Sink stations have no parameters, only incoming connections from one or more stations, depending upon the model.

8. **Class switch station:**
A *class switch* station allows you to define models where jobs are permitted to change class.
The *class switch* component is fully defined by the *class switch matrix*, referred as C.
Let $R = (1, \ldots, r)$ the set of the classes of the system, then C is a $r \times r$ square matrix.
The element $C_{i,j}$, $i, j \leq 0$, is the probability that a job of class i entering this station switches to class j.
More details on `Class Switch` station properties can be found in subsection 3.11.9 - *Class-switch station* of JSIM*graph*.
Set or change of the properties
Double click on the `Class switch` icon to open the `Editing ClassSwitch Properties` station's configuration dialog (see Figure 4.2). There are two sections: `Class Switch Matrix` and `Routing Section`.

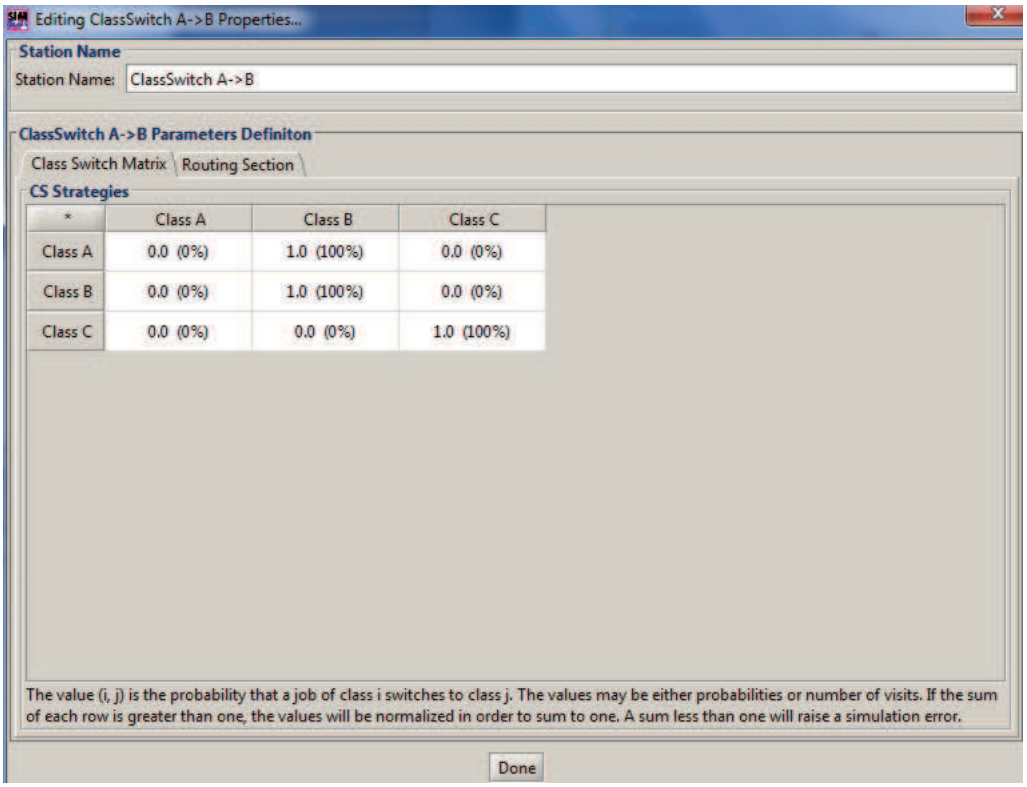


Figure 4.2: Window for the `Class switch Station` configuration.

Class Switch Matrix
In this section you can input the C matrix in order to control how the jobs switch among the classes of the system. Each cell of the matrix may contains either normalized or absolute values, but the sum of each row is constrained to be grater or equals than one.
Routing Section
For each class of customers, the user should select the algorithm that want to apply in order to determine the outgoing connection (i.e., the routing strategy) followed by a customer that completed its service in the current station. The following algorithms are available:
- Random
- Round Robin
- Probabilities
- Join the Shortest Queue (JSQ)
- Shortest Response Time
- Least Utilization
- Fastest Service.

To know details about these algorithms, please refer to the `Routing Section` of subsection 3.11.1-*Source Station*.

9. **Logger station:**
A logging station (i.e., *logger*) reads information flowing through it and writes it to a file. In the simplest way, it is a tool to understand and debug the traffic flow moving through the interesting part(s) of the model. Place the logger station into the model, choose the parameters, and trace the data as it passes through the model.

Set or change of the properties

Double click on the `Logger Station` icon to open the `Editing Logger Properties` station's configuration dialog (see Figure 4.3). There are two sections: `Logger Section` and `Routing Section`.

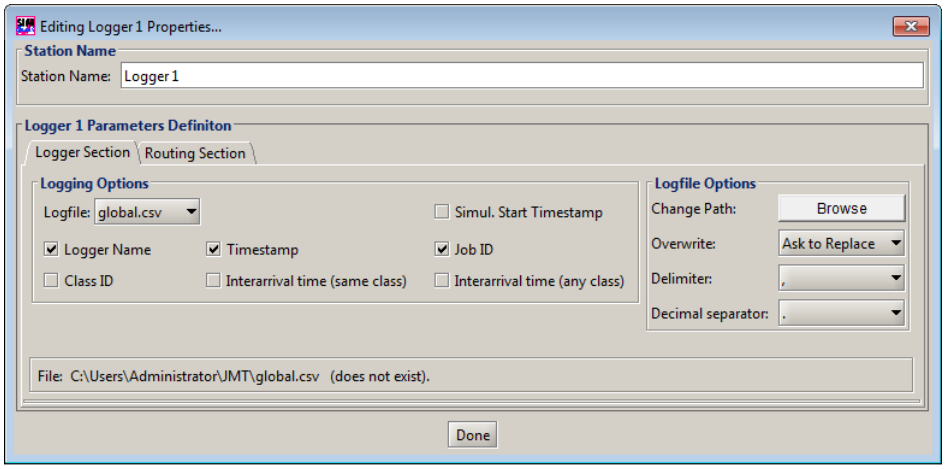


Figure 4.3: Window for the `Logger Station` configuration.

Logger Section

The configuration properties of the `Logger Section` are:

- **Logging Options:**
  - The Logfile's name to use, either individual or merged (Logger0.csv or global.csv, respectively).
  - The information to log (fields): Logger Name, Job Arrival Time (simulation units), unique Job ID, defined Class ID, Interarrival time of same class, Interarrival time of any class. These fields are found in the next section.
- **Logfile Options:**
  - Browse button: allows changing the directory where logfiles are stored.
  - Overwrite method to use when the logfile exists, and a new simulation needs to overwrite the file. Replace overwrites the file, append adds data to the end of the file.
  - Delimiter chooses the character to separate between.
- **Status** displays the path and file-status of the logfile that is going to be written.

As messages flow through the Logger from one station to another, the following information can be logged:

- `Logger Name`, the name of the logger where the message occured (e.g., *Logger0*)
- `Job arrival time`, this timestamp marks the current simulation time from start of simulation (not seconds)
- `Job ID`, the auto-generated unique sequence number of the message (e.g. 1,2,3...).
- `Class ID`, the name of *customer class* of message (e.g., *Class0*)
- `Interarrival time (same class)`, is a time difference between customer of the same class that passes through the logger
- `Interarrival time (any class)`, is a time difference between customers of any class that passes through the logger.

Once a *Logger* is configured, running a simulation produces a logfile with the chosen fields. An example of logfile output is:

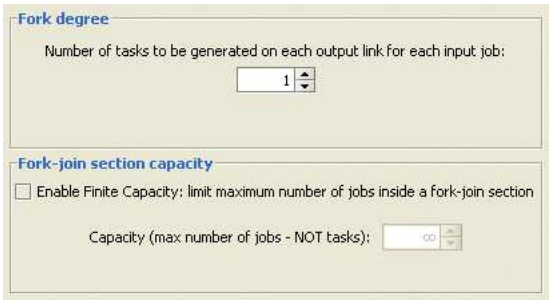
```
LOGGERNAME;TIMESTAMP;JOBID;JOBCLASS;TIMEELAPSED_SAMECLASS; TIMEELAPSED_ANYCLASS
Logger0;0.199;1;Class0;0.000;0.199
```

```
Logger0;0.559;2;Class0;0.341;0.341
```

Fork section

How to define the Forking Behavior This section is characteristic only of fork stations. It defines the parallelism degree of a job, in terms of the number of tasks it is split into and how many jobs can be serviced concurrently.

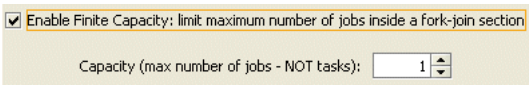
In this section you can define the station forking degree, i.e., the number of tasks created for each job arriving at the fork station, and its capacity, i.e., the maximum number of of jobs that can be in a fork-join section (when a join is present):



Forking degree: it is the number of tasks that are routed on each outgoing link of the fork station. Therefore, each job is split into (forking degree)*(number of outgoing links) tasks. By default the forking degree is 1 and it can be modified using this form:



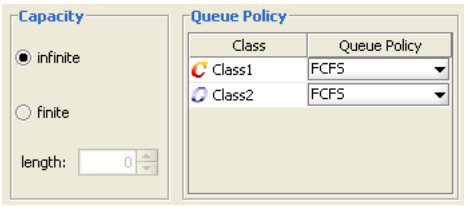
Capacity: it is the maximum number of jobs that can be served by a fork-join section simultaneously. It makes sense only if there is a join station matching the fork one. It can be finite, in which case once the limit is reached, jobs wait in the queue of the fork station. A job is removed from the queue and serviced (i.e., split into tasks) when a job is recomposed at the matching join station and leaves it. Capacity is defined using the form below, after checking the *Finite capacity...* box:



Queue section

How to Define the Queue Strategy
The Queue section is part of the **Station Parameters** definition page; it is present only for server and fork stations.

The Queue section allows the specification of the queueing capacity (whether finite or infinite) and policy. Different classes may have different policies associated with them.



Capacity: a station can accept any customer and let them wait in queue, in which case its capacity is considered infinite, or it can only accept a finite number of customers. In this case its capacity is finite, with a length to be specified in the form 

Queue policy: it is the algorithm used to decide which customer to serve next. A variety of factors can

contribute to the order in which customers are served, such as arrival order, priorities associated with a class, the amount of service already provided to customers, etc.

In JSIM, the main queueing disciplines are based on arrival order and priority and are as follows:

FCFS: Customers are served in the order in which they arrive at the station. If the model is exported to MVA, the following constraint is enforced in the exported model because the MVA analytical algorithm (see the BCMP theorem in [BCMP75]) requires in a multiple class environment the customers of all the classes must have the same average service time $S_{c,k}$ at a FCFS queue station. In order to take care of the differences among the service requirements of the different classes, the total number of visits to the station per each class $V_{c,k}$ is adjusted in order to obtain the correct values of service demands $D_{c,k}$.

FCFS (Priority): Customers are ordered according to their arrival time but customers with higher priority jump ahead of customers with lower priority (a *small* priority number means low priority). Customers with the same priority are served FCFS.

LCFS: An arriving customer jumps ahead of the queue and will be served first. The LCFS discipline implemented in JSIM is not of the preemptive-resume type.

LCFS (Priority): The next customer to be served is one with the highest priority (conventionally a *small* priority number means low priority), so an arriving customer can only jump ahead of the queue of the other customers with equal or smaller priority. Customers with the same priority are served LCFS.

Processor Sharing (PS): A Processor Sharing server works as all the customers are processed simultaneously in the station, but devotes to each of them a fraction of its total capacity inversely proportional to the number of customers. It can be seen as a RR scheduling with a quantum size zero so that the control of the server circulates infinitely rapidly among all the customers. If the station has *multiple servers* (greater than 1), when there are more customers in execution than servers, each server splits its capacity equally across the customers. In the other case, the capacity of a server is completely allocated to the customer it has to execute, so its service time is the same as with FCFS, for example. That is, if a multi-server PS queue has m servers but only $n < m$ resident customers, then, each of them will run inside a single server as-if it was in execution alone while leaving $m - n$ servers inactive. Thus, it will never happen that a customer is executed with a service time less than the time it requires as if it is alone in the station.

Additionally, a number of additional disciplines are also made available to the user:

RAND: Customers are scheduled in random order.

SJF/LJF: Customers are scheduled in shortest (resp. longest) job first order.

SEPT/LEPT: Customers are scheduled in shortest (resp. longest) order of their mean processing time.

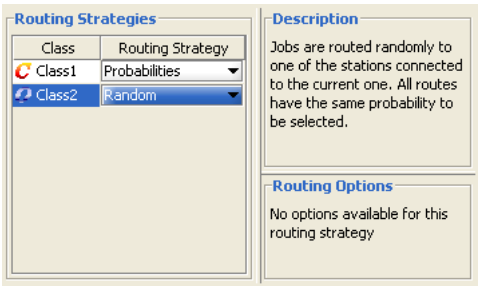
LCFS-PR: In Last Come First Served - Preemptive Resume, customers are processed similarly to LCFS, but arriving jobs with higher (resp. equal or higher) priority preempt the execution of the current job in service, which is resumed later when the server returns idle.

FCFS-PR: Similar to LCFS-PR but customers are processed similarly to FCFS. Hence, arriving jobs of equal priority do not immediately gain access to the server.

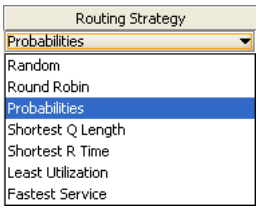
SRPT: Jobs are processing in order of their shortest residual processing time, where the residual processing time is the time to complete service once gaining access to the server.

Routing section

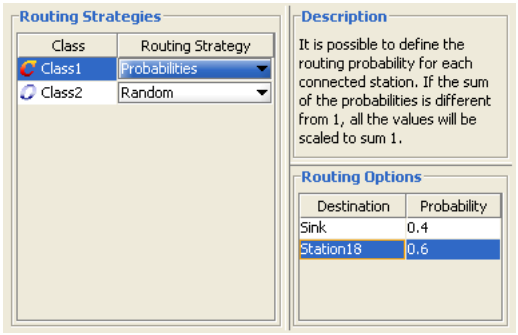
How to Define the Routing Strategy The Routing section is part of the **Station Parameters** definition page defined for all stations, except for fork and sink stations. In the routing section, for every class defined, you may decide how the completed jobs are routed to the other devices connected to station for which the routing strategy is defined.



For each class, the routing algorithm to be used on the outgoing links of the station is selected from this menu:

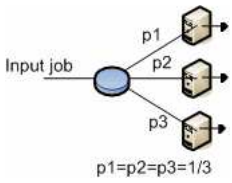


By clicking an algorithm, a brief explanation of the selected algorithm is shown and routing options can be specified,where necessary:

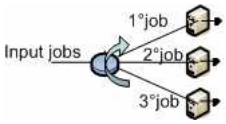


You can choose among the following algorithms (NOTE: in each of the pictures illustrating the algorithms, the blue station implements the routing strategy to the other devices):

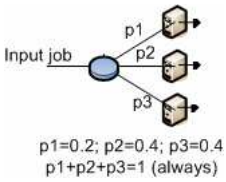
Random: with this strategy, jobs are routed randomly to one of the stations connected to the routing device. The outgoing links are selected with the same probability.The figure illustrates the routing strategy with 3 output links. For each link the probability to be selected is 1/3.



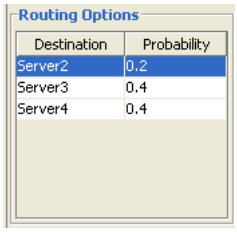
Round Robin: with this algorithm, jobs are cyclically routed to the outgoing links according to a circular routing. As the figure shows, the first job is sent to the top station, the second job is sent to the central station, and the third job is sent to the bottom station. The next job would be sent to the top station again, and so on.



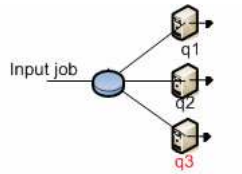
Probabilities: with this algorithm, you can define the routing probability for each outgoing link.The sum of all probabilities must equal 1. If the values provided do not satisfy the constraint, JSIM automatically normalizes the values before the simulation starts.



This strategy requires that you define the probability for each output link via the panel on the bottom right of the window.

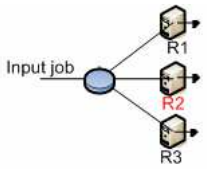


Join the Shortest Queue: with this strategy, each job is routed to the device that has the smallest queue length, i.e., number of jobs waiting, at the time the job leaves the routing station. The figure shows a case where the queue lengths at the devices are 3, 2, and 1 jobs, respectively, from top to bottom. The exiting job will be routed to the bottom station, since its queue is the shortest(1 job).



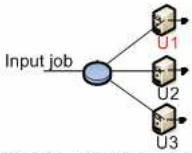
q1=3 jobs, q2=4jobs, q3=1jobs

Shortest Response Time: with this algorithm, jobs are sent to the station where the response time for the job's class is the smallest at the moment a job leaves the routing station.The figure shows that at the time of routing, the middle station has the smallest response time, R, so the job will be sent to it.



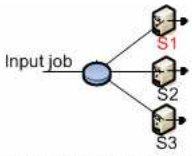
R1=3 sec, R2=2,5 sec, R3=2,9sec

Least Utilization: with this strategy, the destination device is chosen as the one with the smallest utilization at the time the routing is performed. In the example depicted in the picture, the top station is the least utilized, so it will receive the next job to leave the blue station.



U1=34%, U2=44%. U3=45%

Fastest Service: with this strategy, a job is routed to the device with the smallest service time, S, for the job's class. In the figure, the exiting job will be routed to the top station since it service time is the minimum among the three.



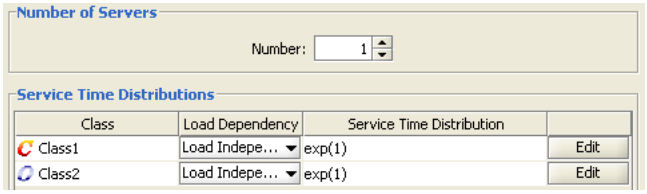
S1=0.34 s, S2=0.89s, S3=1.02s

Service section

How to Define the Service Strategy This section is present in server and delay stations. This section allows the specification of the number of servers, for server stations, and the service time distribution, for both server and delay stations.

Delay stations are infinite servers with identical service time, therefore they only need the service time distribution. The infinite number of servers provides for equal average response time for all jobs, as no job waits in queue for service.

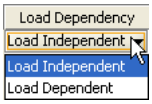
In this section, the load dependent or independent nature of the service time is also specified for each class in server stations:



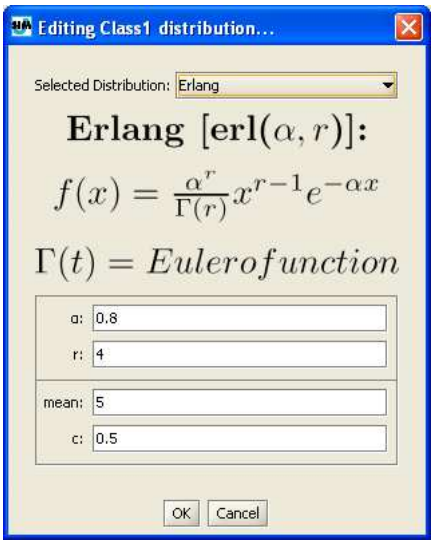
The number of servers in a server station can be modified using the corresponding input area:



For each class, you must specify whether the service time is Load Dependent or Load Independent using this menu:



A load independent service indicates that, regardless of the number of jobs that are in the station, the system will serve all jobs following a fixed policy modelled by the chosen statistical distribution.
For each class, the Service Time Distribution is set to Exponential with average equal to 1. It can be modified by clicking the Edit button and inserting all the required parameters from this window:

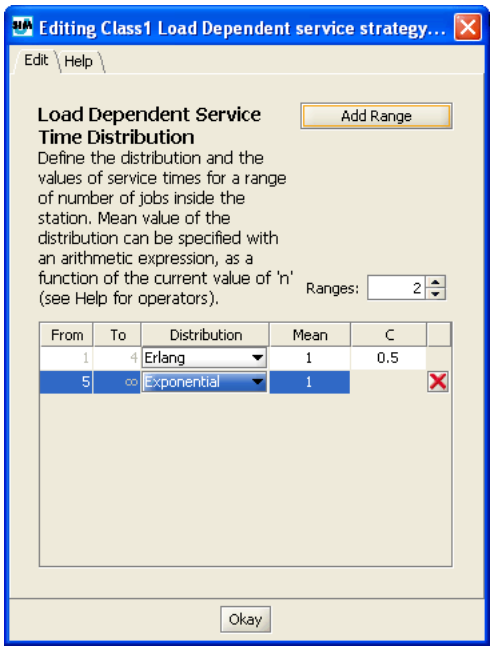


Click the drop-down menu and choose the service time distribution among the following:

- Burst (General)
- Burst (MMPP2)
- Deterministic
- Erlang
- Exponential
- Gamma
- Hyperexponential
- Normal
- Pareto
- Replayer
- Uniform
- Uniform

A load dependent service time indicates that the amount of time the server spends with each customer depends upon the current number of customers in the station. A set of intervals for the number of jobs in the station is specified, either by adding one range at a time via the Add Range button or by specifying the total number at once. Each range must them be specified by its lower (`From`) and upper (`To`) extremes. Each such range can

be associated with different service times, as for the distribution, the mean and the coefficient of variation, or a subset thereof. To set the parameters of a Load Dependent service time, click the Edit button and then specify the parameters for each added range.



For each customer number range you must specify the following parameters:

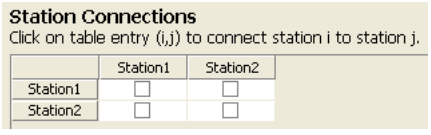
Distribution: you can choose among Deterministic, Erlang, Exponential, Gamma, Hyperexponential, Normal, Pareto, Uniform distributions.

Mean: the mean value of each distribution is specified in the *Mean* form by double clicking on it. Insert a number or an arithmetic expression that will be evaluated with JFEP - Java Fast Expression Parser. For a complete list of the command supported by JFEP you can read the *Help* tab or see the JFEP web site at http://jfep.sourceforge.net/

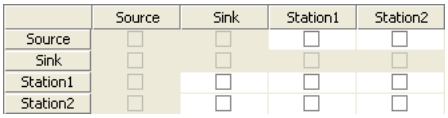
C: the coefficient of variation of each distribution (when C exist) can be specified by double clicking on the *c* form. For example, in the previous picture two policies are defined: From 1 to 4 jobs in the station, the server will behave according to an Erlang distribution with mean = 1 and c=0.5. For any number of jobs greater or equal to 5 in the station, the system will behave according to an Exponential distribution with mean = 1. If you want to delete a range click

4.2.3 Define Connections

The **Connections** tab allows you to define how stations are connected with each other. In order to create a connection from station *i* to a station *j*, check the table entry (i,j) in the connection matrix (rows identify source stations, columns identify destination stations)



NOTE: if there are open classes in the model, the Source and Sink station automatically created by the system appear in the connection matrix. A *Source* station may have only outgoing links, a *Sink* station may have only incoming links.



To define a consistent model a Source station must have at least one outgoing connection and a Sink station must have an incoming connection.

Example 1: if you want to connect station1 to station2 in this way 

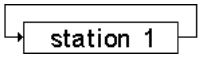
you must check this box

	Station1	Station2
Station1	☐	☑
Station2	☐	☐

Example2: if you want to connect station1 to station2 in this way 

you must check this box

	Station1	Station2
Station1	☐	☐
Station2	☑	☐

Example3: if you want to connect station1 to itself (with a feedback loop) 

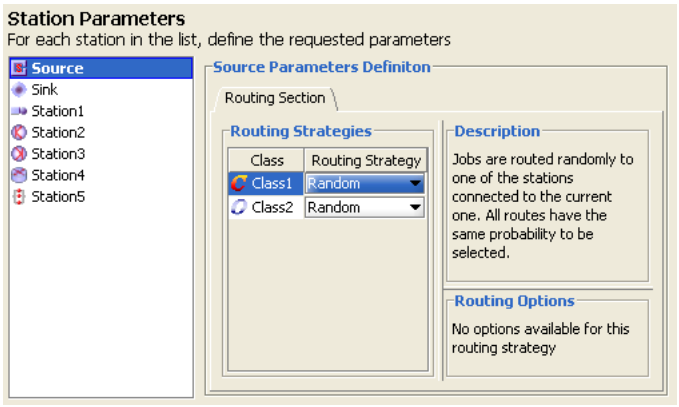
you must check this box

	Station1	Station2
Station1	☑	☐
Station2	☐	☐

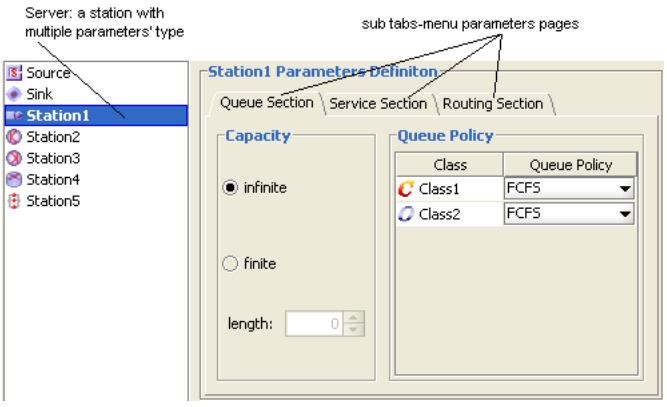
NOTE: if connections among the stations are inconsistent, the system displays error-warning messages. For more information about connection errors, read ref.

4.2.4 Station Parameters

The **Station Parameters** tab allows you to define the characteristic parameters of each station. For each station, a different menu is presented depending upon the station type. Queuing, service, routing and forking, or subsets thereof, are the characteristics to be defined through a series of tabs. Default settings are provided that can be modified at the user's need. The Sink station does not have any parameter.



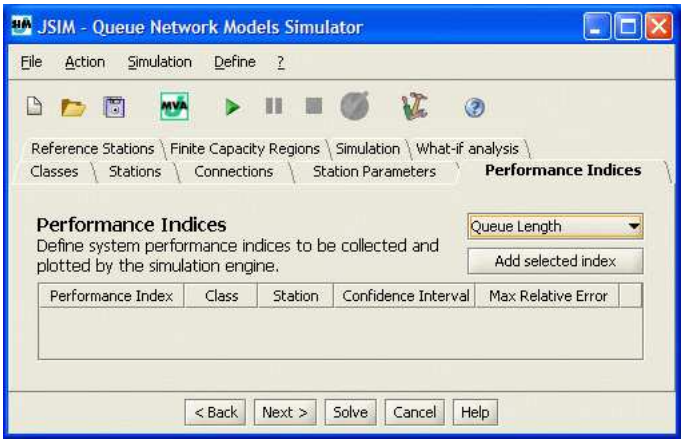
Example (with a server station):



NOTE: You can find detailed information about station parameters subsection 4.2.2.

4.2.5 Define Performance Indices

Performance Indices are selected using the **Performance Indices** tab.



You can choose any subset of indices from the list below to be plotted as the model output:

Number of Customers Number of customers at a station, both waiting and receiving service.

Queue Time Average time spent by the customers waiting in a station queue. It does not include the Service Time.

Residence Time Total time spent at a station by a customer, both queueing and receiving service, considering all the visits at the station.

Response Time Average time spent at a station by a customer for a single request (it is the sum of Queueing time and Service time).

Response Time per Sink Average time spent in system by a customer before dropping at the selected sink.

Utilization Percentage of time a resource is used w.r.t. the system lifetime; it varies from 0 (0%), when the station is always idle, to a maximum of 1 (100%), when the station is constantly busy serving customers for the entire system lifetime. Utilization may be greater than 1 if a station has more than one server.

Throughput Rate at which customers departs from a station, i.e., the number of services completed in a time unit.

Throughput per Sink rate at which customers departs from the system with respect to the selected sink, i.e., the number of requests completed in a time unit that reach a given sink.

System Response Time Average time a customer spends in the system in order to receive service from the various stations it visits. It corresponds to the intuitive notion of response time, as the interval between the submission of a request and the reception of the response.

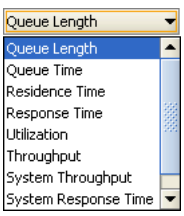
System Throughput Rate at which customers departs from the system.

System Number of Customers Average number of customers in the system. If the index is associated with a closed class, then it is equal to the number of customers in the class.

System Power The optimal operational point of a system is the point corresponding to the maximum System Throughput X with the minimum System Response time R, i.e., the value of the ratio X/R is maximized in this point. This ratio is known as *System Power* .

Each index is associated with a class and a station and will be computed within a given Confidence Interval and Max Relative Error, both defined on the (0-1) range, by performing the following steps:

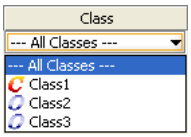
1. Select the index you want to add to the model from this menu:



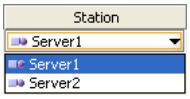
2. Click 

and the index will be added to the panel. Next the index parameters must be set.

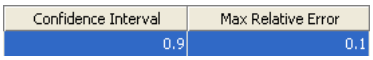
3. Select from the Class menu a single class, or All Classes, for which the index must be computed.



4. Select the Station for which the index must be computed from Station menu. In case of system wide indices, namely, System Throughput and System Response Time, this option is not available.



5. Double click on the values to modify the default values for the Confidence Interval size of the solution and for the Max Relative Error of the greatest sample error, if you want more/less accurate results.



6. Repeat these steps for all the indices you want to include in the model output.

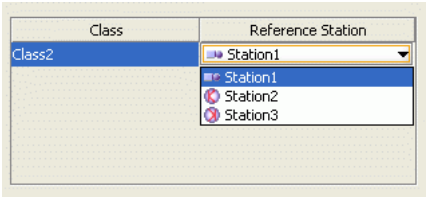
NOTE: erroneous parameter settings will be detected only when the simulation is started, raising a warning or an error message.

4.2.6 Reference Stations

The **Reference Stations** tab allows you to specify the Reference Station for each customer class. For each customer class, a reference station must be specified in order to compute system throughput for that class. For open classes, there is only one possible reference station, that is, the Source station. Such an association is done by the system automatically each time an open class is created and cannot be modified.



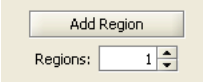
For each closed class, any of the existing stations can be selected using the scroll menu below



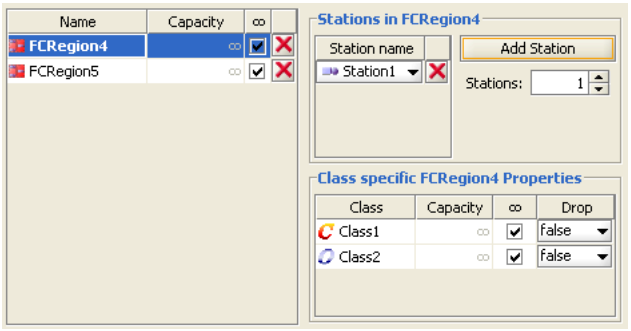
NOTE: if a reference station is missing for any of the classes, an error message is returned when the simulation will start.

4.2.7 Finite Capacity Regions

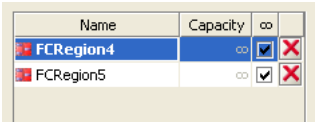
The **Finite Capacity Regions** tab allows you to define Finite Capacity Regions in the model. Finite Capacity Regions can either be added one at a time by clicking the *Add Region* button or all at once by selecting the desired number in the *Regions* selector.



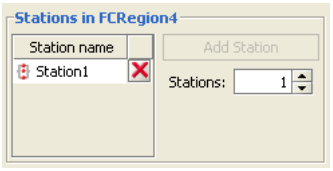
When a new region is created, two new sub areas devoted to the parameters of the region, namely the stations involved and class properties when in the region, appear on the screen on the right.



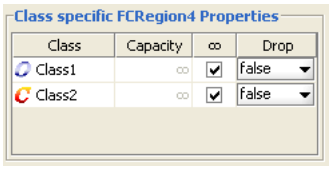
For each region, define first the capacity (infinite by default). If the check in the Infinity box is removed, the desidered number can be input in the Capacity field.



Then for each region define which stations are part of that region. Stations are added in alphabetical order on their names. If the model has more than one station, a drop-down menu from the station name allows you to change the station to add.

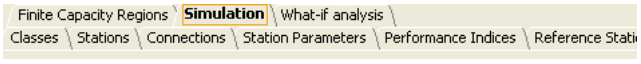


Finally, in the bottom right panel, you can define class specific properties for the Finite Capacity Region selected. Region capacity for that class (finite or infinite -the default value) and the dropping characteristic (true or false) in case the capacity is exceeded are defined here.



4.2.8 Define Simulation Parameters

The Simulation Parameters and Initial State of the model are defined using the **Simulation** tab.



Simulation parameters: In the top section of this windows you can edit general simulation parameters.

Simulation Seed: The simulation seed is a number used by the simulation engine to generate pseudo-random numbers. As this numbers are pseudo-random, if you change the seed, you will obtain a different sequence of pseudo-random numbers. If a simulation is repeated using the same seed, the same sequence of pseudo-random numbers will be generated, thus leading to identical results. The default value is random, which indicates that the simulation engine will pick a seed of its choice in a pseudo-random fashion each time it is started; uncheck *random* and insert a number if you want a custom simulation.

Maximum duration (sec): It represents the maximum amount of time in seconds that the simulation will run. If the simulation ends before the maximum duration, the parameter is ignored and does not affect the results. The default value is *infinite* deselect it and specify the preferred maximum time if you do not want the simulation to run for a possibly very long time. In this case, the simulation stops when the time limit is reached, although a solution may not be available yet.

Maximum number of samples: It is the greatest number of samples for each index that JSIM collects before ending the simulation. During a simulation, measurements can be stopped in case of:

- Success, if the results have reached the required Confidence Interval and the Max Relative Error
- Failure, if the simulation has analyzed the maximum number of samples but has not reached the required Confidence Interval or Max Relative Error
- Failure, if timeout occurs before successfully calculating the final results.

The default value for the maximum number of samples is 500,000; you may increase it (for a more accurate simulation) o decrease it (for a faster simulation).

Representation Interval (sec): This is the granularity at which results are plotted on the screen, i.e., the time interval before a new point is added to the graphs, as the simulation proceeds. A large value will make the simulation to proceed slower, as the graphs are updated less often. Small values will provide an impression of better *responsiveness* from the simulation, as graphs are updated more frequently.

Initial state: The initial state is the model situation at time 0, typically described by the number of customers present in each station at the beginning of the simulation. For closed classes, all customers are allocated by default to their reference station. This arrangement can be modified as long as the total number of jobs remains the one defined in the Classes (subsection 4.2.1) tab. For open classes, it is possible to preload each station with any desired number of customers. The following table represents an example of the initial state of a model with three stations and two classes.

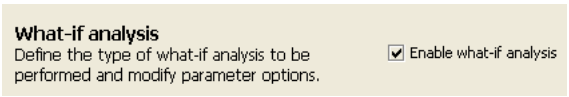
	Station1	Station2	Station3
Class1 (Ni = 6)	6	0	0
Class2	0	0	0

4.2.9 Define what if analysis

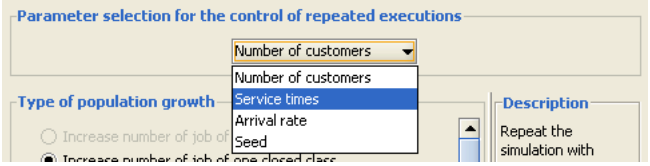
The parameters of a what-if analysis are defined using the **What-if-analysis** tab.



A What-If Analysis consists of a series of simulations in which one or more parameters are varied over a specified range. This allows the observation of system behavior under a spectrum of conditions, unlike the single JSIM simulation run where the system is observed under a specific set of configuration parameters. By default the what-if-analysis is not enabled. It must be activated explicitly and its parameters defined. **Activating the What-If Analysis**
After completing the definition of the model and selecting the **What-if analysis** tab, check the Enable what-if analysis checkbox to activate it.



Selecting the What-If Analysis Parameter
After enabling the what-if analysis, it is possible to select the parameter to control the series of simulation runs using the menu below:

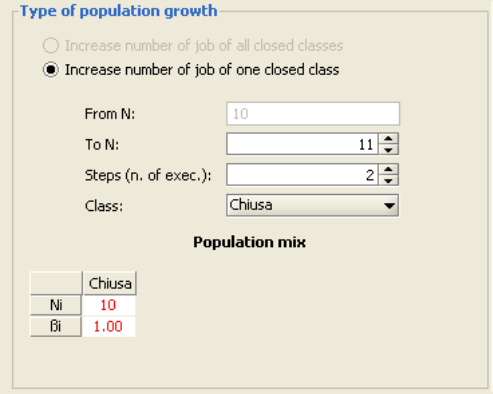


When you select a what-if analysis parameter, the bottom section of the window will change depending upon the parameter. In the right portion a description is provided of the selected parameter and of the impact of its variation. In the left portion the details of the parameter range (From and To fields) and the number of executions (Steps) on the range are provided. The set of modifiable parameters depends upon the number and type of classes in the model:

- In a single class model, you simply select the parameters you want to change during simulation.

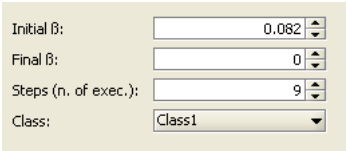
- In a multiclass model, you select the parameter and specify whether you want to apply the variation to all classes or just to one class.

Number of Customers (only for models with closed classes) JSIM repeats the simulation changing the number of jobs in each run, starting from the number inserted in the *From N* field, to the value inserted in the *To N* field. The simulation is repeated *Steps (no. of exec.)* number of times.



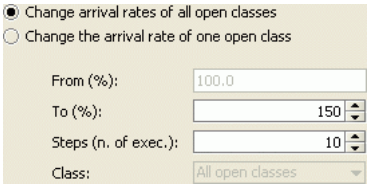
In the figure above a what-if analysis is planned on a single class model. The initial value of the number of customers is not modifiable from this window, as it is part of the **Classes** tab. The final value is specified in the field *To N*. In this case, with a final value of 20 and 6 Steps, simulations are run for 10, 12, 14, 16, 18 and 20 customers, respectively. Only the customer number in the class selected in the *Class* (Chiusa, in the picture) will be changed. The remaining classes will keep their initial number of customers. The simulator makes sure that the sum of the percentages of customers in various classes add up to 1, as shown in the *Population mix* table on the bottom of the window. If the *all-classes* option is selected, the overall population is increased while keeping the relative proportion of jobs in the various classes constant. Because the number of customers in each class can only be an integer number, only the population vectors with integer components can be considered. Therefore, the actual number of executions may be smaller than the one specified.

Population Mix: (only for models with two closed classes) The population mix describes the way the population is divided between the two class, i.e., the percentage of customers in each class over the total population. Because of the complexity of the underlying modelling, this type of analysis is possible only for models with two closed classes. Mixed class models can be analyzed too, as long as there are two closed classes. Only the closed classes will be considered.



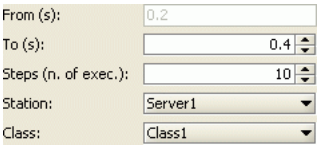
In order to allow for a complete analysis of the population mix, in this case the initial value of the percentage of customers in one of the two classes, the one selected through the **Class** field, can be modified. Its default value is the ratio of the class population to the total population of closed customers, defined in the **Class** tab. Both the final and the initial *beta* are numbers on the range [0-1].

Arrival Rate: (only if there are open classes) Arrival rate is the frequency at which jobs arrive at a station during a period of time. Similarly to the number of customers, the arrival rate can be changed for one specific open class or for all the open classes in the model. If a single class is selected, the final arrival rate and the number of steps must be specified. The initial arrival rate is specified in the Arrival Rate section of the *Classes* tab and it is not modifiable here. If the arrival rate is to be changed for all classes, the final value is expressed as a percentage, which is applied to all classes.



The figure above shows the settings for a what-if analysis on the arrival rate of all the open classes. The *To* field is seto to 150%, which means that for each class the final arrival rate will be one and a half times the initial value. 10 runs will be exectued with equally spaced (in percentage) intermediate values.

Service time (for all types of classes) Service Time is the time required by a customer class at each visit at a station. In this case, besides the final value and the number of runs to be executed, the station and the customer class (or all the classes) whose service time will be varied must be specified.



The figure above shows the settings for a what-if analysis of the service time of class Class1 at station Server1. The service time distribution will not change. Only its average will be modified to span the range defined by the initial value (specified in the **Station Parameters** tab) and the final value specified here in the *To* field. If the *All classes* option is selected, the range of service time to explore is expressed in percentage, starting from the initial value specified at model definition (100%). The station where the service time change is applied must be specified.

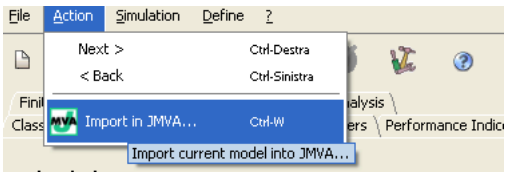
Seed (for all types of classes)



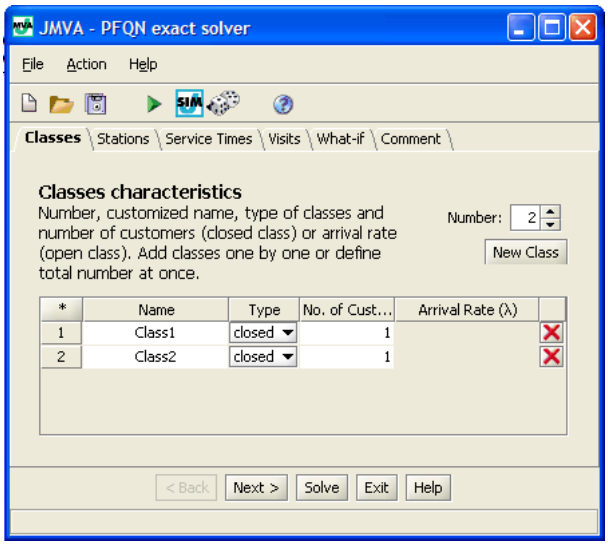
Unlike the previous cases, where the analysis is performed for a range of values of one of the model parameters in order to investigate the system behavior under a variety of conditions, a what-if analysis on the seed aims at evaluating the sensitivity of the simulation engine to numerical conditions, in particular to the initial value fed to the pseudo-random number generator, i.e., the seed. The Simulation engine uses a pseudo-random number generator to generate the sequence of values used in each execution. By changing the seed, a different sequence of values is produced, thus leading to different numerical results. The first value is the one defined in the **Simulation Parameters** tab. In this panel, the number of executions is specified and the simulation engine generates as many pseudo-random seeds to be used in the executions.

4.3 Import in JMVA

After creating a simulation model, you can export it to the analytic solver component of the JMT suite, namely JMVA. In the Action menu, click *Export to MVA*.



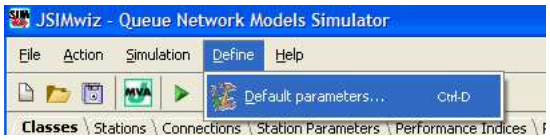
If there are errors in the model, i.e., conditions not allowed in models with analytical solution, a dialog window with information about such errors appears (See section 4.6). If you want to continue with the analytical solution, you must corret the errors. When the model satisfies all the constraints for analytical solution, a JMVA window appears on the screen and the model can be solved with JMVA.



Refer to JMVA user's guide (chapter 2) for specific help on JMVA functionalities.

4.4 Modify default parameters

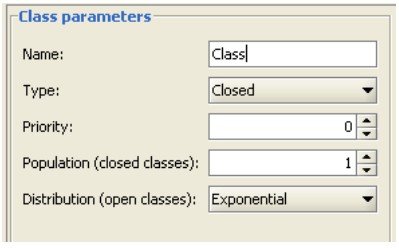
The simulator default settings as for Class, Station, Simulation, and Finite Capacity Region parameters, may be changed using the `Define` menu in the menu bar. Click the `Default parameters`



or the  *Define default values of model parameters* button. In JSIM all parameters have predefined, or default, values. Such values can be change to suit the user's most common modelling activities. Defaults can be modified for the following sections:

- `Class Parameters`
- `Station Parameters`
- `Simulation Parameters`
- `Finite Capacity Region (FCR) Parameters`

Class Parameters These settings define the default class parameters used when a new class is created. For detailed information about classes, see subsection 4.2.1.



Name: the default name for each newly created class of customers is *Class*, followed by a progressive integer, starting from 1, indicating how many classes you have created so far. It can be reset to any alphanumerical string, which will be followed by the same numbering scheme.

Type: the default class type is Closed. If open classes are used more often, you may want to change to Open as default, so as to minimize the type changes in the class definition section.

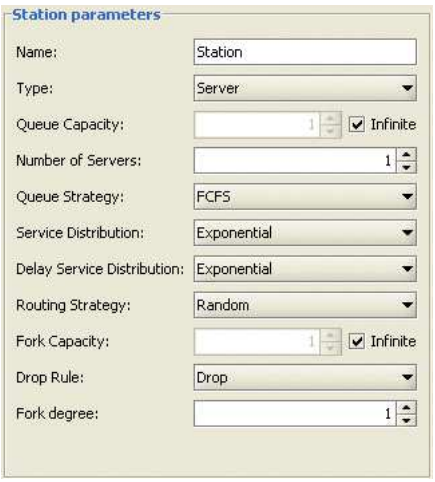
Priority: the default class priority is 0. Higher numbers correspond to higher priority.

Population: this parameter applies only to *closed classes*; the default value is 1, i.e., all closed classes are

created with 1 customer in each of them. Change the value if you want larger populations by default in each closed class.

Distribution: this parameter applies only to open classes; the default value is *Exponential* since it is the most popular distribution that characterizes customer interarrival times at a system. Change it to any of the distributions available if most of your customer interarrival times follow a non-exponential pattern.

Station Parameters These settings are general parameters for every type of station. For a detailed description of station types, see subsection 4.2.2



Name: the default name of each newly created station is `Station` followed by an increasing integer, starting from 1, indicating how many stations have been created so far. It can be reset to any alphanumerical string, which will be followed by the same numbering scheme.

Type: the default station type is `Queue`, since this is the most popular type of station in performance models. Possible type of stations are `Delay, Queue, Fork, Join, Router, Logger, ClassSwitch`.

Queue Capacity: this parameter applies only to `Queue, Fork` stations; the default value is infinite, i.e., an infinite number of customers can queue in such stations. It can be reset to finite, with a finite value specified, by checking out the `Infinite` check box.

Number of Servers: this parameter applies only to `Queue` stations; the default value is 1, i.e., single server. It can be changed to any finite value .

Queue Strategy: this parameter applies only to `Queue` and `Fork` station types; the default value is FCFS and can be changed to LCFS, RAND, LJF, etc. A priority among the job classes can be specified with all these disciplines.

Service Distribution: this parameter applies only to `Queue` stations; the default value is Exponential, as it is the most popular when modeling device service time. It can be changed to any of the available distributions if most of the devices modeled are non-exponential.

Delay Service Distribution: this parameter applies only to `Delay` stations; the default value is `Exponential` but i can be changed to any of the distributions available .

Routing Strategy: this parameter applies only to `Delay, Queue, Join, Logger, Router, ClassSwitch`; the default value is `Random` and it can be changed to any of the available routing strategies, namely `Random, Round Robin, Probabilities, Join the Shortest Queue, Shortest Response Time, Least Utilization, Fastest Service, Load Dependent Routing`.

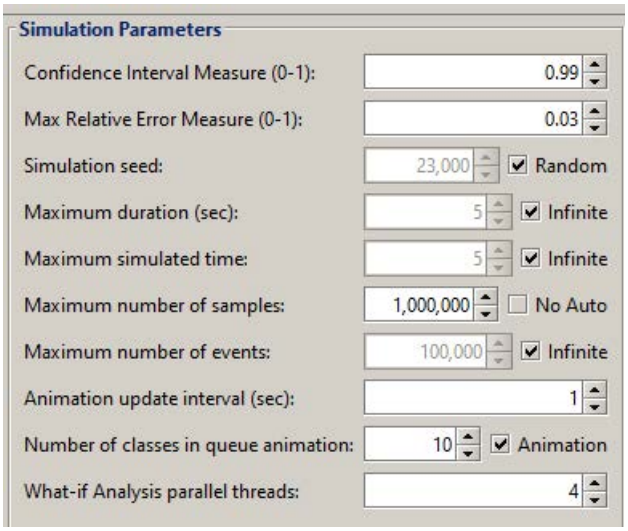
Drop Rule: this parameter applies only to `Queue, Fork` station types. When a finite capacity is defined, the default value of the Drop Rule is `Drop`, i.e., all customers exceeding the station capacity are discarded. Two other policies are possible: `Waiting Queue, BAS Blocking`. The *BAS, Blocking After Service* block the customers in the station before the one considered if its maximum capacity is reached. BAS implements the

FBFU policy, *First Blocked First Unblocked*, which means that each target queue keeps a separate list of the BAS blocked queue and when there is empty space unblocks the first in the list. When the capacity is infinite, the Drop Rule is disabled by default.

Fork Capacity: this parameter applies only to `Fork` stations; the default value is infinite, i.e., there is no limit on the number of customers inside a *Fork-Join* section of the network. It can be changed to finite, by checking the `Enable Finite Capacity` checkbox and specifying a finite number.

Forking Degree: this parameter applies only to `Fork` stations; the default value is 1, i.e., one task is generated for each outgoing link of the *Fork* station for each input customer. It can be changed to any finite value, thus increasing the number of tasks each customer is split into on each parallel link.

Simulation Parameters These parameters define the simulation behavior from a statistical point of view. For more information, see subsection 4.2.8 for simulation parameters or subsection 4.2.8 for Confidence Interval and Max Relative Error.



`Confidence Interval Measure:` this parameter is set by default at 99%, a commonly used value as it provides a good trade-off between results accuracy and simulation time; larger values will lead to more accurate results at the expenses of an increase in the time the simulation takes to complete, smaller values will achieve the opposite effect.

`Max Relative Error Measure:` this parameter is set by default at 3%; smaller values will increase results accuracy while bigger values allow for larger errors in the samples. This metric is intended as the maximum ratio between the half-width of the confidence interval and the estimated mean.

`Simulation Seed:` this parameter is used to initialize the pseudo-random number generator; the generated sequence of pseudo-random numbers is correlated with the seed used. A `Random` value is selected by default, or a user may define its own value.

`Maximum Duration:` this parameter is set to infinite by default, so that even lengthy simulations can complete and satisfy even narrow confidence intervals; finite values (in seconds) may force termination before the end of the simulation.

`Maximum Simulated time:` this parameter is set to `Infinite` by default, this is the time simulated during the simulation run. As a function of the objective of the study, a user may want to stop the simulation at a given time. In this case, the user may uncheck the `Infinite` check box and type the desired value of the simulated time.

`Maximum Number of Samples:` this parameter is set by default to 10,000,000; with a smaller number of samples it may not be possible to achieve the requested confidence interval, a larger number may not necessarily increase the results accuracy and simply extend the simulation time.

`Animation Update Interval:` this parameter indicates the time interval between consecutive graph updates on the screen; the default value is 1 sec, as it provides for a smooth progression of the graphs on the screen. Larger values will make the evolution of the simulation appear jerky.

`Number of classes in queue animation:` this parameter controls the number of classes for which graphs will be plotted for the selected performance indexes; by setting the default value to 10, all classes are represented in most models. Graph animation is enabled by default but can be disabled.

`What-if Analysis parallel threads:` this parameter sets the number of threads that will be used to independently simulate JSIM models during a what-if analysis. The default value is set automatically by JMT to be the number of logical cores of the machine (in the figure: 4), thus if hyper-threading is enabled this value will be larger than if hyper-threading is disabled. Once the default value is manually modified, the same setting will be used again when JSIM is re-started. A user may decide to set this value to a lower number of threads to make sure that the machine is responsive and can smoothly execute other applications while JMT runs a what-if.

Finite Capacity Region (FCR) Parameters These parameters define the default values for newly created Finite Capacity regions. For information about Finite Capacity Regions, see subsection 4.2.1

`Name:` the default name of each newly created Finite Capacity Region is `FCRegion`, followed by a progressive integer, starting from 1, indicating how many classes you have created so far. It can be reset to any alphanumerical string, which will be followed by the same numbering scheme.

`Global Region Capacity:` the default value of this parameter is infinite, i.e., there is no upper limit to the total number of customers allowed in the region; the value can be changed to finite, and a number must be specified.

`Region Capacity per Class:` the default value of this parameter is infinite, i.e., there is no upper limit to the per class total number of customers allowed in the region; the value can be changed to finite and a number must be specified.

`Drop:` the default value of this parameter is `False`, i.e., no customer is ever discarded when arriving at the region; it can be changed to `True`, in which case customers in excess of the region capacity are discarded.

4.5 Modify the Current Model

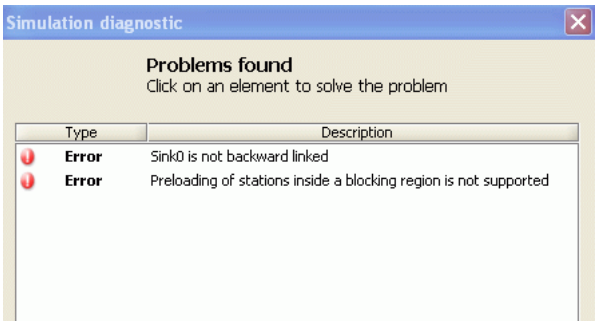
After creating a model and possibly running a simulation, you can go back to the model and modify it. To modify the current model parameters after a simulation, close the simulation windows with the  button and select the tab of the feature you wish to modify. No constraints on the order apply in this case, since all the model parameters are already defined. If no simulation was run, simply select the tab of interest.



Parameters are changed the same way they were set when creating the model the first time section 4.2.

4.6 Error and warning messages

When you start the simulation, JSIM analyzes the model and if there are errors or inconsistencies, a diagnostic window will pop up, describing the problems found.



In what follows, a list of common problems and the respective solutions are presented:

- **Error** Source0 is not forward linked

  This error occurs when the station (source or join) has not outgoing, or forward, links to any other station in the model. Only Sink stations do not have forward links. To see how to connect two stations, see subsection 4.2.3.
- **Error** Sink0 is not backward linked

  Each sink station must have at least an incoming, or backward, link from other stations, for open class jobs to be able to leave the system. Only the Source station does not have a backward link. To see how to connect stations, see subsection 4.2.3.
- **Error** No reference station defined for Class4

  For each class you must define a Reference Station that is used to compute the performance indices for the class. To see how to define a reference station, see subsection 4.2.6.
- **Error** No performance indices defined

  This error occurs when you try to start the simulation but no performance index is defined. Performance indices are defined through subsection 4.2.5.
- *Warning* Fork found but no join

  Error Join without fork

  Fork and Join stations should be inserted together in the model. If you have inserted only one of the two stations, when the simulation is run JSIM will show a diagnostic message. Having a Fork without a Join is possible, although it may lead the system to saturate quickly, hence the Warning message. Having a Join without a Fork is a mistake, hence the Error message. The missing station must be added with the corresponding links, see subsection subsection 4.2.2 and subsection subsection 4.2.3 respectively.
- **Error** No classes defined

  Customer classes are a mandatory parameter of a model. If you forget to define any, the simulator will raise an error. Add classes as described in subsection 4.2.1.
- **Error** A performance index is defined more than once

  In the definition of the model output, the same index has been added more than once for the same station and class. Multiple occurrences must be removed, see subsection subsection 4.2.5.
- **Error** Close class Class1 routed to station Server0 linked only to sink

  Customers of closed classes keep circulating in the system. They may not visit a Sink station or a station that is connected on the outgoing link only to the Sink, since this would cause them to leave the system. The station connections must be changed, see subsection 4.2.3.
- "Round Robin" routing strategy in Class0 for Source is not allowed. This was considered as RandomRouting

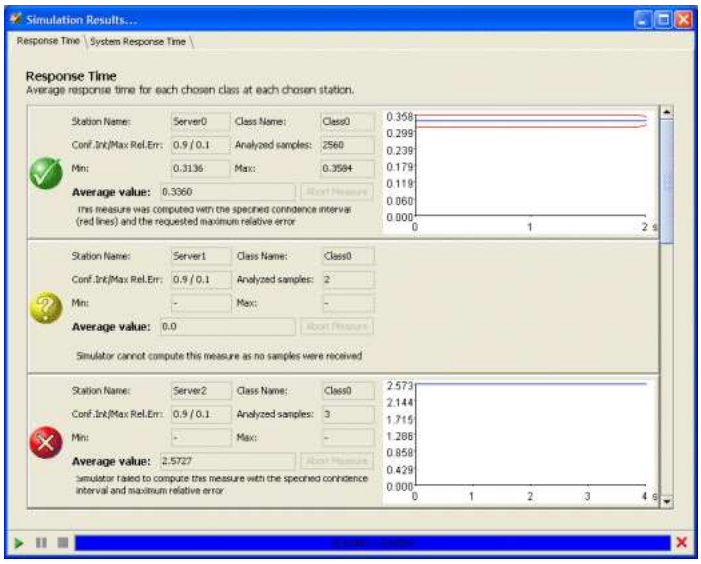
  When exporting a JSIM model to JMVA, a few constraints apply as for the admissible routing strategies. Round Robin routing is not implemented in JMVA, so it will be automatically transformed into Random, if you click the *Continue* button. Otherwise, you can change it as described in subsetion subsection 4.2.4.
- *Warning* Different service times inside FCFS station Server0

  When exporting a JSIM model to JMVA, a few constraints apply as for the admissible service time distributions. A FCFS station must have exponentially distributed service time with the same mean for all the classes of customers that visit the station. If you click the *Continue* button, the station service time is reset to the default value, the same for all classes. Otherwise, you can change it as described in subsection 4.2.4.
- **Error** Undefined station in performance index

  When you add a new performance index, the station for which it is defined must be specified, otherwise the error message above appears. Specify the missing station as described in subsection subsection 4.2.5.

4.7 Simulation Results

When a simulation terminates successfully, performance indices are plotted on tabbed windows, as illustrated below.

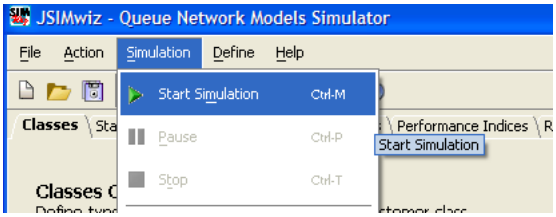


In each window, results for all the stations for which the performance index is specified are listed. Numerical values and graphs are given. Successful results are indicated by a checkmark sign on a green bullet. In case of errors or of early termination, a cross on a red bullet indicates that the results obtained are not within the requested confidence interval. In case of syntactically correct components that make no sense from a modelling point of view, e.g., a station that is never visited by any customer, a question mark on a yellow bullet indicates that the simulator could not compute the index for lack of measurements. Values are available for:

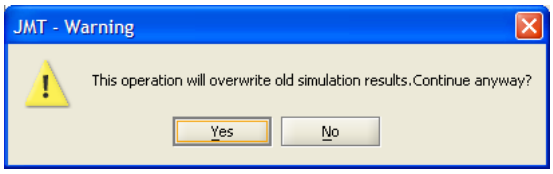
- Station Name: This is the name of the station for which the index is computed.
- Class Name: This is the name of the class for which the index is computed at the station (it could be *All*, to mean All Classes).
- Conf.In / Max Rel.Err.: This is the Confidence Interval and Maximum Relative Error of the computed index.
- Analyzed Samples: This is the number of samples used to compute the performance index.
- Min.: This is the minimum value observed for the index.
- Max.: This is the maximum value observed for the index.
- Average Value: This is the computed average value of the index, usually the value of greater interest.

4.8 Start Simulation

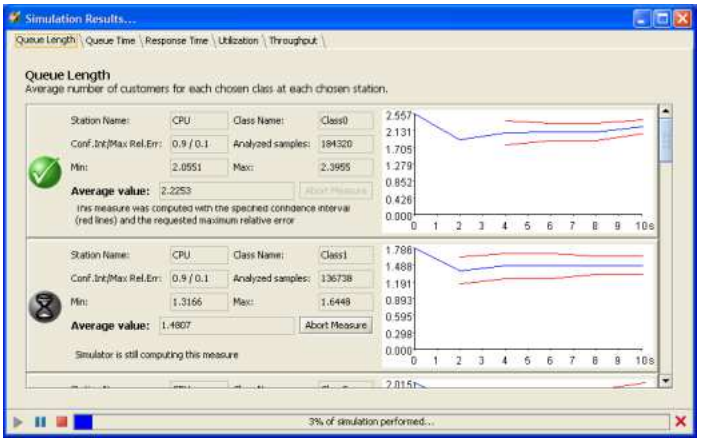
When a model is complete and the Simulation Parameters have been defined, you can start the simulation clicking the  button or selecting *Start* Simulation from the Simulation menu



If there are errors and/or potentially critical conditions, error and/or warning messages will appear, see section 4.6. In case of errors, the simulation may not start and errors must be corrected. In case of warnings, you can still start the simulation but the results may not be consistent. If another simulation was executed before the current one, JSIM will warn you that previous simulation results will be overwritten if you continue. Click Yes if you do not care about previous results, click No if you wish to save previous results first and then restart the simulation.



After the simulation has started, the results window appears, with partial values. In particular, for each performance index and for each station the index has been selected, the current number of analyzed samples, the current average value and the graph of the latter are constantly updated. The graph plots in blue the current average value estimated by the simulator and in red the confidence intervals. Confidence intervals appear in the graph when the simulator has gather sufficient samples to compute a reasonably accurate confidence interval: note that the number of samples required for the confidence intervals to appear is not constant and depends also on other factors such as the degree of fluctuation of the average value estimate.



An hourglass indicates that the computation of that index is not finished yet. A check mark in a green bullet indicates that the computation of that index has successfully terminated. In this case, the minimum and maximum average are reported as well. At the bottom of the result window are the simulation control buttons, namely the pause button (double bar), the start button (right pointing triangle), and the stop button (square).The status bar indicates the percentage of simulation executed so far, both graphically and numerically.

Chapter 5

JMCH (Modelling Classroom Helper)

5.1 Introduction

JMCH is a tool developed to enhance the teaching activity of performance evaluation topics with the support of animation technique. The current version is focused on the animation of *five scheduling algorithms* (FCFS First Come First Served, LCFS Last Come First Served, SJF Shortest Job First, LJF Longest Job First, PS Processor Sharing) of a single queue station, of *three routing algorithms* (RR Round Robin, JSQ Join Shortest Queue, Probabilities) of a network with three queue stations, and of the *Markov Chain states* of a queue station with different configurations.
Through a user-friendly graphical interface it is possible to interact with the simulated algorithms during their execution. Using the single-step animation feature, you can compare the behavior of different algorithms at each action.
The JMCH home screen (see Fig.5.1) allows the selection of the algorithm to be considered .

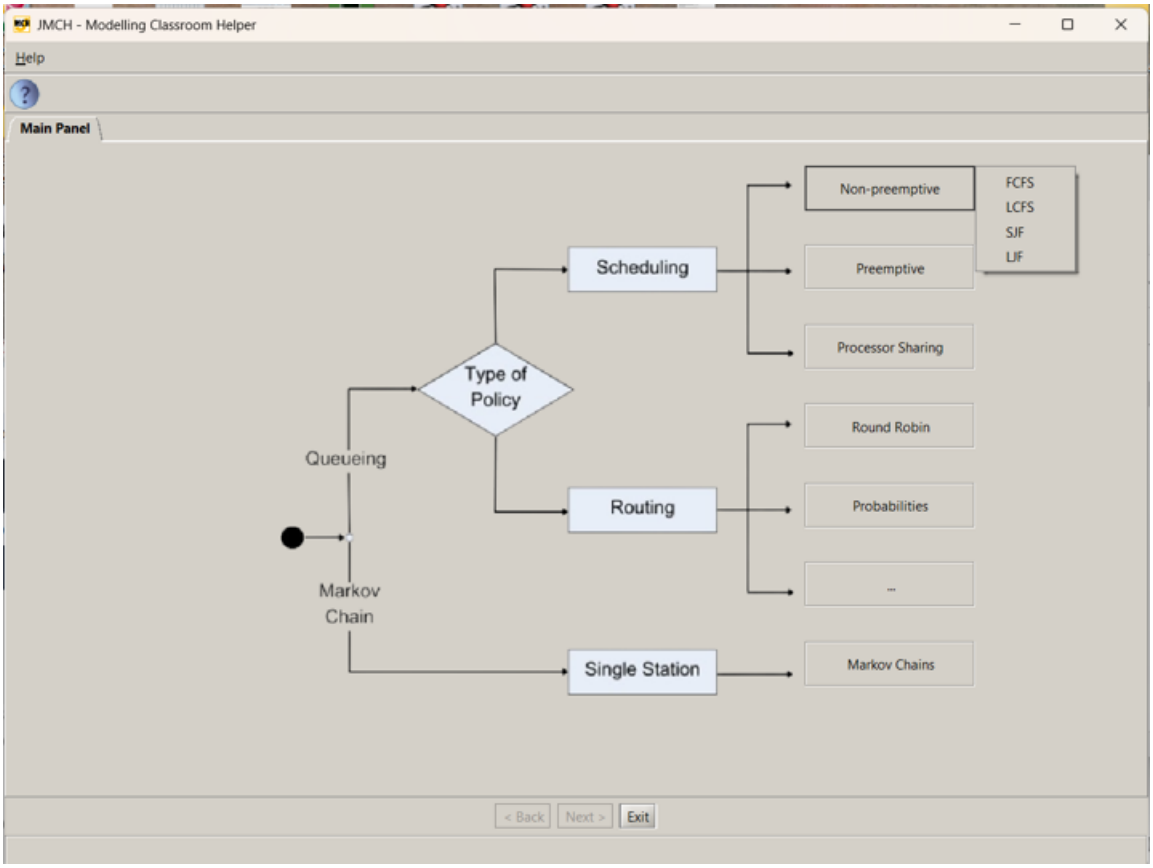


Figure 5.1: The JMCH home screen for the selection of the algorithm to be analyzed

5.2 The Scheduling algorithms

We consider a single queue station with one or two servers. The capacity of the queue of jobs waiting for the execution is *limited to five*. Jobs arriving when the queue is full are *dropped*. Thus, the *maximum capacity* of the station is 5 jobs in the queue plus the jobs that are in execution (one for each server). In the following, for simplicity, we will consider only one server.
The five scheduling algorithms simulated are four non-preemptive and one preemptive:
— **First-Come First-Served** (FCFS): jobs are executed according to their arrival order at the station
— **Last-Come First-Served** (LCFS): jobs are executed in the opposite order of arrival at the station
— **Shortest Job First** (SJF): the job in the queue with the shortest service time is served first, jobs are ordered based on this policy
— **Longest Job First** (LJF): the job in the queue with the longest service time is served first, jobs are ordered based on this policy
— **Processor Sharing** (PS): all jobs arrived are immediately served but the fraction of server capacity assigned to each of them depends on their number.

When the simulation of a scheduling algorithm is selected, the Simulation Panel of Fig.5.2 is opened.

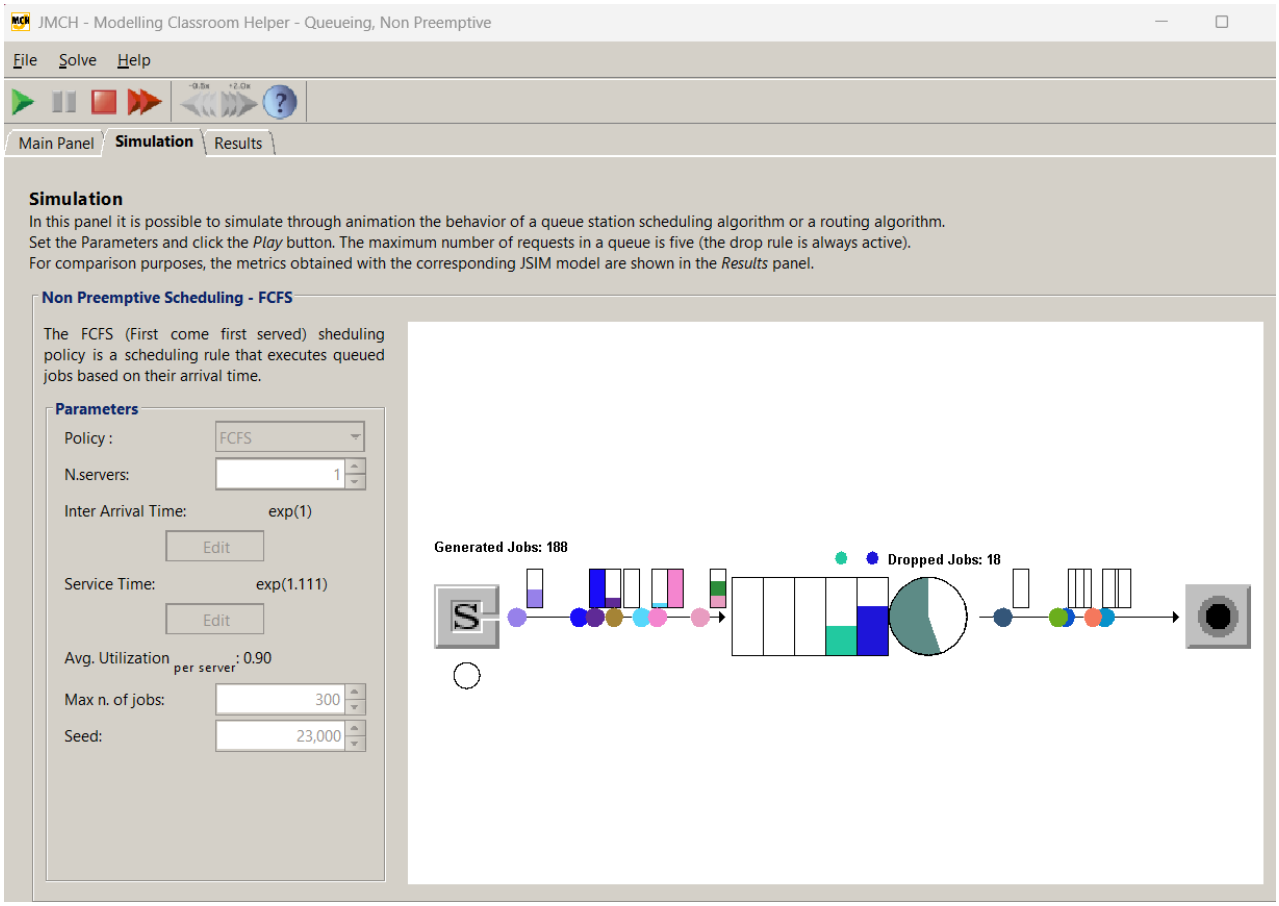


Figure 5.2: The `Simulation panel` of scheduling algorithms

The graphical animation takes place in this panel which includes the following components:
• `Menu & Button Toolbar`: this gives users the ability to control the flow of the animation with various options
• `Parameters section`: describes the model parameters that can be modified by the users
• `Graphical section`: where the model is displayed, the animation takes place once it starts.

The description of the three sections of the panel follows.

Menu & Button Toolbar section
The buttons of this section (see Fig.5.3) are utilized to control the flow of the simulation. There are seven buttons:
• `Play Button`: start the animation; is enabled only when the animation is paused or has not yet started.



Figure 5.3: The `Menu & Button` toolbar of the `simulation panel` section for the control of the graphical animation.

• `Pause Button`: pauses the animation. It is active only when the animation is currently playing. At each `Pause`, the performance metrics obtained up to that point are calculated and printed in the `Results` table. Therefore, if the simulation is restarted by pressing the `Start` button again, these values will be overwritten in the table by the values obtained at the end of the new simulation, either because either because it was completed or interrupted by pressing the `Pause` button again.
• `Restart Button`: restart the animation using the same parameters set in the `Parameters` Section. This button is always enabled.
• `Next Step Button`: execute only one step of the animation. When this button is pressed, both the `Play` and `Pause` buttons are turned off. The animation will stop when a new event occurs, such as a new job arrived in the model, a job start or complete its execution.
• `Increase/Decrease Velocity Buttons`: control the speed of the animation. These buttons can be pressed in runtime, so they are only functional when the simulation is running. The user can double the simulation's speed (x2.0) or reduce it by half (x0.5).
• `Help button`

Parameters section
In this section the `Parameters` of the variables used in the simulation can be defined (see Fig.5.4).
— `Policy`: the policies implemented are: FCFS, LCFS, SJF, LJF (non preemptive) and PS (preemptive)

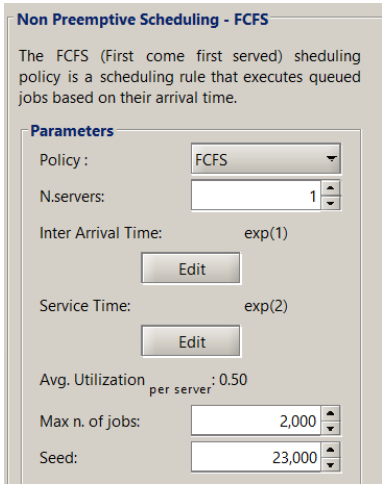


Figure 5.4: The Parameters setting section of the `Simulation panel` for the FCFS scheduling algorithm.

— `N.servers`: the number of servers in the queue station considered (one or two)
— `Interarrival Times`: the correspondent `Edit` button allows the setting of the distribution of the interarrival times of the jobs arriving at the station (`Exponential, Uniform, Deterministic, Hyperexponential`) and their mean value (or their arrival rate λ). The default values are `Exponential 1`.
— `Service Times`: the correspondent `Edit` button allows the setting of the distribution of the service times of the jobs (`Exponential, Uniform, Deterministic, Hyperexponential`) and their mean value S (or their service rate). The default values are (`Exponential 1`).
— `Utilization per Server`: displays the utilization of each server, computed as: $U = (\lambda S)/n$
where `n` is the number of servers associated with the station (1 or 2). If is U is greater than 1, the station is in saturation and the execution of the model cannot start.
— `Max. Number of job`: specifies the maximum number of jobs generated by the source that arrive at the station (the simulation will stop when this value is reached).
— `Seed`: the user may define a seed for the random number generator. The default value is 23000. This feature may be helpful when you want to compare the behavior of different algorithms using the step-by-step feature. For example, to analyze the difference between FCFS and LCFS, it is possible to open two separate JMCH

windows: one running an FCFS simulation and the other an LCFS simulation. By advancing the simulations step-by-step, it becomes easier to observe the differences between the two policies in real time.

Graphical section
This section shows the graphical representation with animation of the model (see Fig.5.2 and 5.6). A model for the simulation of the scheduling algorithms consists of the following components:
• *Source*: generate the arriving jobs according to the interarrival distribution defined in the Parameters Section. The number of jobs generated is shown online during the animation. The interarrival time of each job is displayed as a pie chart filled with the color associated with the job, the colored fraction decreases over time and reaches zero when the job is generated. Each job is identified by a circle filled with the same color used in generating its interarrival time. Additionally, a rectangular shape with a fraction of the same color of the job represents its service time (see Fig.5.5).
• *Sink*: collects the jobs that exit the model when their execution is completed. The dropped jobs are not



Figure 5.5: The identifiers of a job moving along an edge.

considered here.
• *Queue Station*: consists of:
— A *finite queue* with *five* slots, represented by a rectangle divided into *five* sections. The jobs in this queue are waiting that the server is free and execute them. Thus the maximum number of jobs in the station is five plus one if there is one sever or two if there are two servers. Jobs arriving when the the five slots of the queue are busy are *dropped*. The number of dropped jobs are shown online during the animation.
— *One or two servers*, shown as circles, where jobs are processed.

With *Processor Sharing* scheduling algorithm, jobs do not wait for idle server in order to start their execution. The server handles all jobs in the station (i.e, in queue and in the server) simultaneously, allocating each job a fraction of its processing power. If no jobs are waiting in the queue, the job currently being processed utilizes the full computational power of the server. However, if multiple jobs are present, each one receives a fraction of the processing power (i.e., the full capacity divided by the number of actual jobs in the system). The queue length may range between 0 and 5 jobs, while the number of servers ranges from 1 to 2. When a job arrives at a busy server, it is still placed in the queue. However, unlike previous configurations, it does not wait for the server to finish processing the current job. Instead, it starts processing immediately, sharing server resources equally with other jobs that are already running. Visually, the color in the rectangles representing the service time will gradually decrease in height as they are being processed. When a color's height reaches zero, the execution of the job is completed and is removed from the station.
The distance between consecutive jobs moving on the arrival and exit paths of the queue station corresponds to their interarrival and inter-exit times. The arriving jobs are placed in the queue according to the scheduling policy selected. For example, Fig.5.6 is a screenshot of the jobs scheduled according to the SJF Shortest Job First policy. As can be seen, the four jobs in the queue are in increasing order according to the policy.

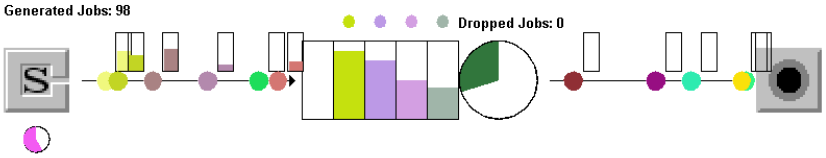


Figure 5.6: Screenshot of jobs scheduled according to the SJF policy.

5.3 The Routing algorithms

For routing algorithms the simulation follows a similar structure of the one described for the scheduling algorithms. In this case the system used for the animated graphic simulation is a network of three queue stations , each with limited capacity of five jobs in the queue and one server. A new component, the *router*, is added between the source and the queue stations. The router's task is to determine the destination of each incoming

job by selecting one of the three possible outgoing edges, based on the policy specified by the user. The arriving time at the `Router` is considered the arriving time at the system. Since its service time is zero, and the time spent in the path is null, this same value coincides with the time when the job reaches one of the queues.
Three routing algorithms are simulated: `Round Robin (RR)`, `Probabilities`, and `Join the Shortest Queue (JSQ)`.

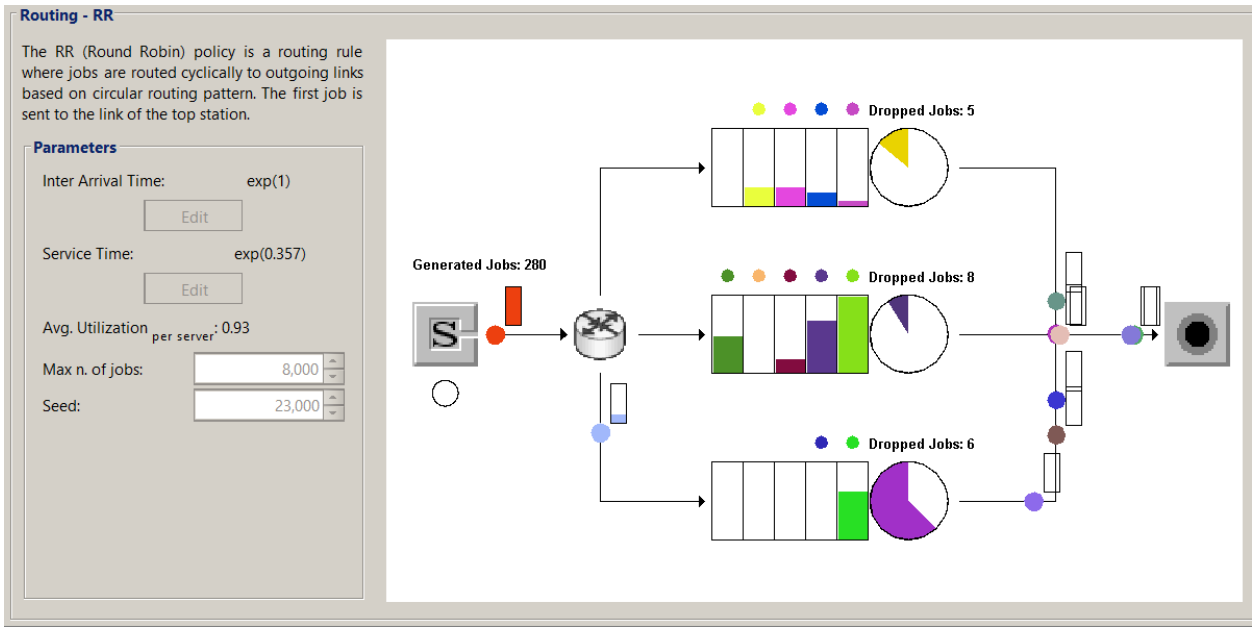


Figure 5.7: The `Simulation panel` of the routing algorithm `Round Robin` (RR)

— `Round Robin (RR)`: the jobs are assigned to the three paths in a sequential order. So, the first job arriving is sent to *Path1*, the second to *Path2*, the third to *Path3*, the fourth to *Path1*, and so on.
—`Probabilities`: Each path leaving the router has its corresponding probability, as specified in the `Parameters Section`, displayed next to it. This helps the user clearly see the likelihood of a job being routed to each of the three queue stations. P1 is the probability of the first path outgoing from the router, P2 is the probability of the second path outgoing from the router, and P3 is computed by default as: P3=1-P1-P2 . If the sum of the probabilities exceeds 1, the `Play` button will remain disabled.
— `Join the Shortest Queue (JSQ)`: an arriving job is routed to the queue station with the minimum number of jobs waiting for execution. To give a clearer picture of the decision-making process, an indicator is shown above each station, displaying the queue size of that station. This way, users can easily understand how the policy routes jobs based on queue length. Sometimes it may happen that `Queue Size` displays a different number of processes in the queue than the one currently displayed in the figure. The motivation is as follows. When a job is sent by the router to one of the outgoing paths, it should be added immediately to the queue of the station in the path. However, for clarity in the graphical animation, jobs are visually moved along the paths toward the stations, allowing users to better understand where a job is moving after being routed. Despite this visual movement, the job is actually queued as soon as it is routed and the time spent on the link is null. So, for example, a job can be accounted in the `Queue size` of a station even if visually it is still travelling towards the station.

When observing a graphical animation of the *Routing Policies*, the user can notice a difference in speed between the jobs routed to the central station and those routed to the upper and bottom stations. This speed adjustment simulates the requirement that jobs must take the same amount of time to reach their destination once routed. Since the paths have different lengths, the speeds were adjusted accordingly

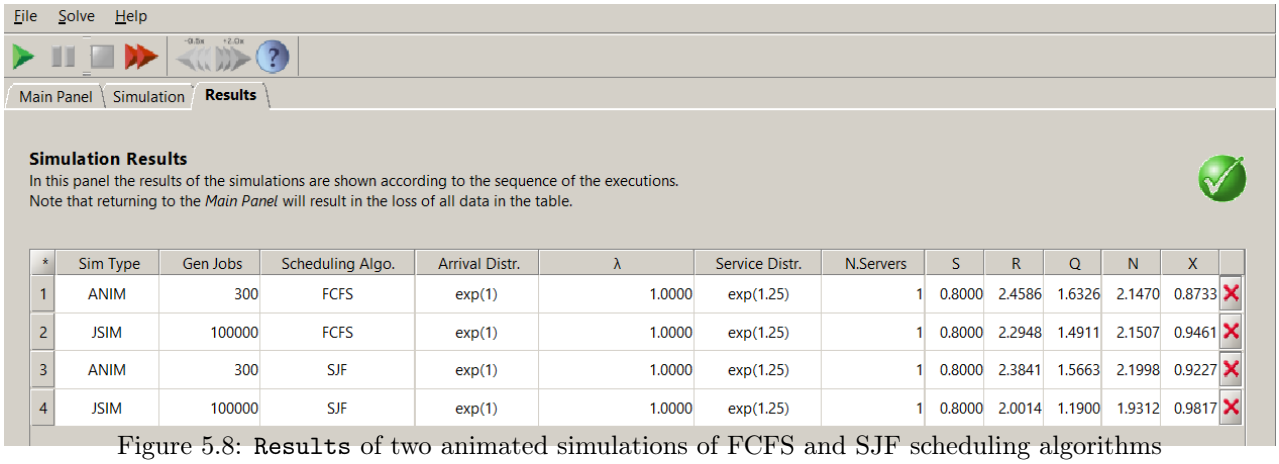
5.4 The graphical animation mechanism

When the `Play` button is pressed, the user sees all the graphical animations in action. To create graphical animations, since JMT uses the Java Swing library, the approach takes advantage of the capabilities of this library.
In general, graphical systems work by repainting graphical components each frame per second while simultaneously updating some of the component's parameters. For example, if we want to move a circle in an application, the circle would have fields for its x and y coordinates within the window. The `repaint()` method would then

be called for each frame, displaying the circle in the position defined by x and y. If x and y change each frame, for instance, by incrementing x by 1 each time, the circle would appear to be moving. In reality, we aren't moving the circle itself but merely updating its coordinates and repainting it in each frame. This is the basic concept behind graphics systems. The goal was to add this behavior to the tool as well. A new thread called `Animator` was created to handle the animation. When a control button is pressed, triggering a new state in the control button's state chart, certain actions are applied to this thread. For instance, clicking the `Pause` button would pause the `Animator` thread, effectively stopping the animation. The basic behavior of this thread is to continuously call the `refresh()` method of the animation class, simulating the effect of refreshing the application at each frame per second.
Each component in the animation is represented by a class that extends *Java Swing's JComponent* class. *JComponent* serves as the base class for all *Swing* components and is used as a foundation for custom components. By extending this class, each custom component inherits the following methods:
• `paint(Graphics g)`: used by Swing to draw the component.
• `repaint()`: triggers another call to the paint() method.
These custom components are then contained within a main `animation` class.
Now let's examine what happens when the `Play` button is pressed. The system switches to a multi-threaded environment, where:
• A primary process is responsible for running the application. This process is the same one that runs when a basic Java program is executed.
• The `Animator` thread is instantiated at this moment and begins running.
The `Animator` thread repeatedly calls the `refresh()` method of the associated animation. This method updates all the parameters for each component (each one has its own logic based on what a single component has to do), and it invokes repaint() for each *JComponent* within the animation.

5.5 Results of the Animated simulations

The second panel that appears in the wizard when the user selects the type of animation from the `Main Panel` is the `Results Panel` (see Fig.5.8). This panel presents the numerical results obtained from the graphical simulations of JMCH and the ones obtained with a corresponding model obtained with JSIMg with the parameters described in the *Simulation Panel*. Usually the results obtained with JMCH show some differences compared to



*	Sim Type	Gen Jobs	Scheduling Algo.	Arrival Distr.	λ	Service Distr.	N.Servers	S	R	Q	N	X
1	ANIM	300	FCFS	exp(1)	1.0000	exp(1.25)	1	0.8000	2.4586	1.6326	2.1470	0.8733
2	JSIM	100000	FCFS	exp(1)	1.0000	exp(1.25)	1	0.8000	2.2948	1.4911	2.1507	0.9461
3	ANIM	300	SJF	exp(1)	1.0000	exp(1.25)	1	0.8000	2.3841	1.5663	2.1998	0.9227
4	JSIM	100000	SJF	exp(1)	1.0000	exp(1.25)	1	0.8000	2.0014	1.1900	1.9312	0.9817

Figure 5.8: `Results` of two animated simulations of FCFS and SJF scheduling algorithms

those obtained with JSIMg, even if the same seed of the two random number generators are used. The reason is that JSIMg results are computed when confidence intervals are met, while JMCH simulations are stopped when the user-selected maximum number of jobs is reached (and no confidence intervals are calculated).
Every row in the results table describes the performance indices of a new simulation run. The values of a animated simulation (ANIM) are computed and shown in the table every time a `Pause` button is pressed or when the simulation ended. New values are overwritten on previous ones.
The metrics that are shown for *Scheduling Algorithms* and *Routing Policies* simulations are different.
The indices shown for the *Scheduling Algorithms* are:
• `Sim Type (Simulation Type)`: indicates the type of simulation run, with the animation of JMCH (ANIM) or with the simulator JSIMg (JSIM).
• `Gen Jobs (Generated Jobs)`: indicates the number of jobs considered to compute the values of the metrics.
• `Scheduling Algorithm`: indicates the scheduling algorithm selected for the simulation (`FCFS, LCFS, SJF,`

LJF, PS).
• Arrival Distribution: indicates the distribution selected in the Parameters section of the Simulation Panel for the inter-arrival time of jobs.
• λ **Arrival rate**: shows the arrival rate of the jobs in the simulation, as selected in the *Parameters section* of the Simulation Panel.
• Service Distribution: displays the distribution chosen in the *Parameters section* of the Simulation Panel for the service time of jobs.
• N.Servers: indicates the number of servers (1 or 2) at the single station of the model, chosen in the Parameters section of the Simulation Panel.
• **S (Service Time)**: shows the mean of the service time distribution, as selected in the *Parameters section* of the Simulation Panel.
• R (Response Time): displays the average response time, computed with the values obtained from the simulation run.
• Q (Queue Time): indicates the average queue time, computed with the values obtained from the simulation run.
• N (Number of Customers): shows the average number of customers, computed with the values obtained from the simulation run.
• X (Throughput): displays the average throughput, computed with the values obtained from the simulation run.
The results of the *Routing Algorithms* includes some minor changes:
• Routing Algorithm: indicates the routing algorithm selected for the simulation (RR, Probabilistic, JSQ).
• N.Queues: shows the number of queue stations in the system. In this version of JMT is fixed at 3.
• Additional fields such as Arrival and Service distributions, R, N, and X have "Sys" as prefix to emphasize that these parameters are calculated not for just one station, but for the *entire system*.

5.6 Markov Chain animation

JMCH provides a graphical representation of the states assumed by a Markov Chain generated by a single queue station model, including single-server station with finite (M/M/1/k) or infinite (M/M/1) queue size, and multiple-server station with unlimited (M/M/c) or limited (M/M/c/k) queue size. It is possible to change the arrival rate λ and service time S of the station during the simulation run time and, if the station has limited capacity c, the size k of the queue (buffer) can also be changed dynamically.
A graphical user interface interface with animation has been implemented. Users may have the visual perception of the bursts of traffic and of their effects on the queue length and on the utilization of the server(s). The exact analytical results are computed and are shown for comparison purposes.

5.6.1 Starting the simulation

Selecting the  button on the starting screen, the dialog window for the selection of the type of station to be considered is shown (see Figure 5.9).

The type of station can be modified later by using the menu item Queue - Change station type .

Four station types are simulated (see Figure 5.10):

- M/M/1: 1 server and infinite queue size.
- M/M/1/k: 1 server and finite queue size k. The queue size k can be set dynamically while the simulation is running. The minimum value of k is 2.
- M/M/c: c servers (homogeneous) and infinite queue size. The number c of the servers should be set with this dialog box and it cannot be changed dynamically while the simulation is running.
- M/M/c/k: c servers (homogeneous) and *finite* queue size k. The number c of the servers should be set with this dialog box and it cannot be changed dynamically while the simulation is running. The *queue size* k can be set dynamically while the simulation is running. The *minimum value* of k is c+1.

When a multiple server station is selected, the text field for the number c of servers is activated.

When the station type is set, the window of Figure 5.9 is shown.

In the main view user can check the *type* of station selected as seen from the *title* bar. The menu bar consists of: Queue, Setting and Help. In the Queue menu user can change the type of the station. In the Setting menu user can set the colors of the user interface and also size of the icons. To start the Simulation user needs to press Start button, and the dialog of Figure 5.11 is displayed. The options displayed are

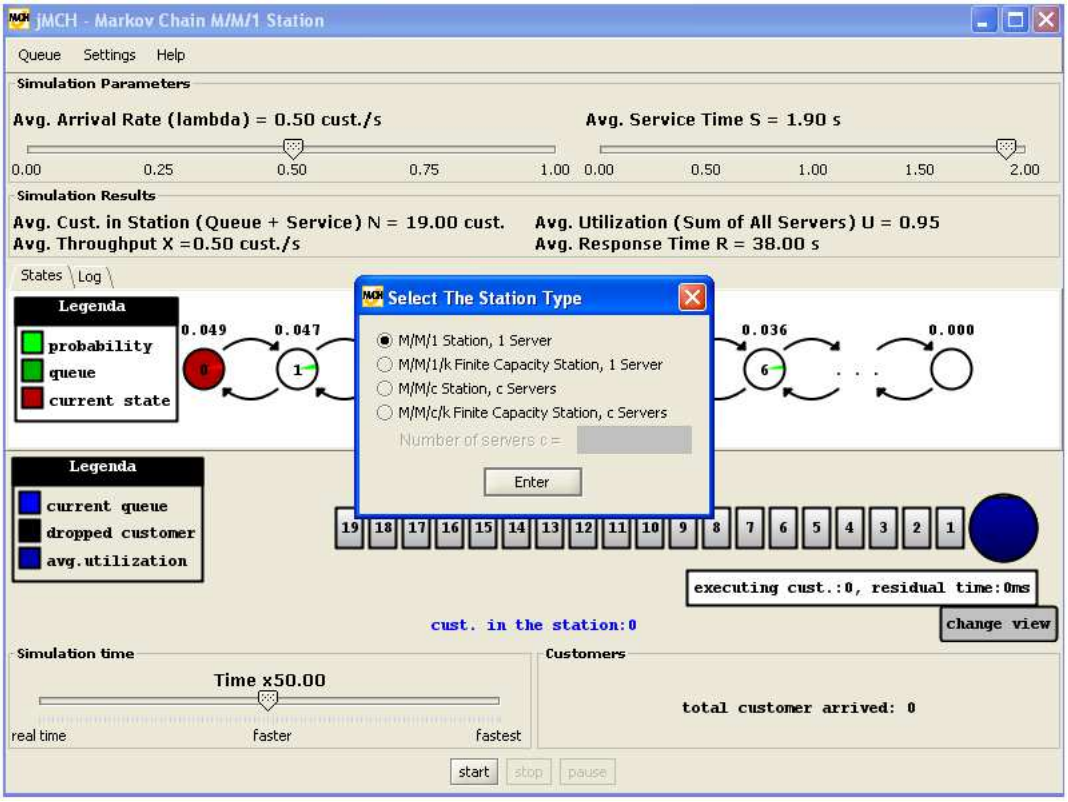


Figure 5.9: The Markov chain animation screen

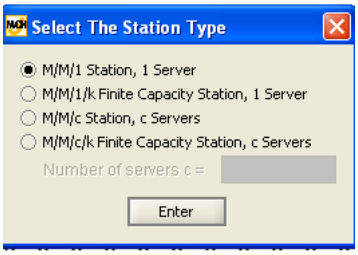


Figure 5.10: Selection of station type

- `Unlimited`
- `Limited(Type in)`

If users select `unlimited` the simulation must be stopped manually else it runs forever. In this case the animation of the queue length and of the Markov chain states can be observed more clearly. If users select `limited`, the simulation runs for the given number of customers. Then, the simulation starts running, as can be seen in Figure 5.12

5.6.2 Input parameters

The top most pane known as *Simulation Parameters* Figure 5.12, is for setting simulation parameters:

- λ: Average arrival rate [customers per second], the distribution of interarrival times is exponential with mean $1/\lambda$.
- S: Average service time for each customer [seconds], the values are exponentially distributed
- k: Maximum station size: is the maximum number of customers allowed in the station. This is the total number of customers including those in queue and those in the server(s). The minimum value of k is 2.

According to the parameters selected (type of station, number of servers, max capacity, arrival rate, service times) the following performance indices are analytically computed and displayed:

- *mean number of customers* in the station (either in queue and in the servers)

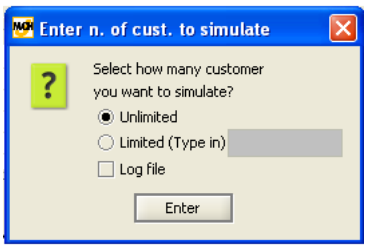


Figure 5.11: Selection of the `Number of Customers` of a simulation

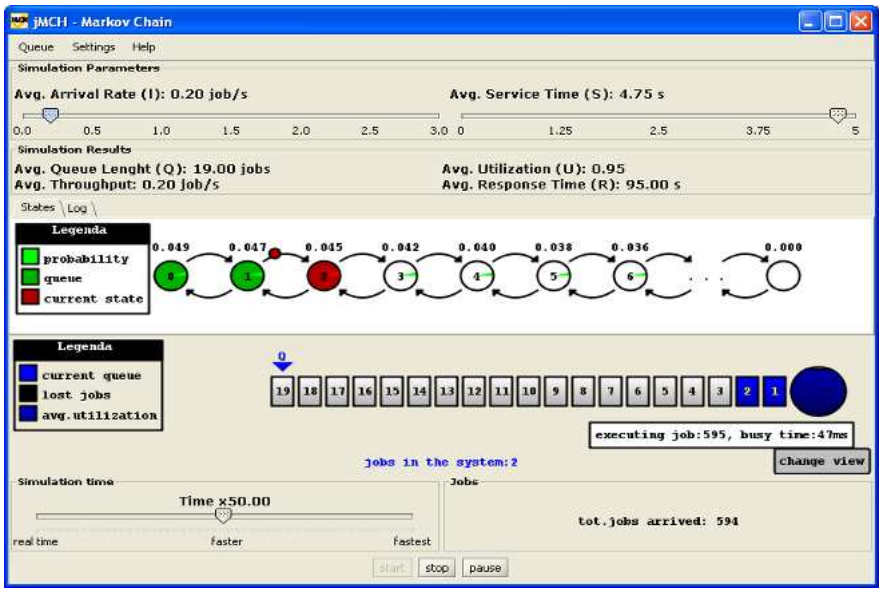


Figure 5.12: Example of display during a Simulation

- *throughput* (that coincides with the arrival rate)
- *utilization*
- *mean response time*
- *probability of the states* of the Markov Chain.

5.6.3 Output results

The pane *Simulation Results* consists of two possible views `States` and `Log`. The first view, called `States` displays the probabilities of the states (analytically computed) and the migration among them with animation obtained through simulation.
The second view is the `Log` screen which shows the data, explained as below, for each customer:

- *Cust.ID*: the unique `id` assigned to each customer by a counter.
- *Arrival Time*: shows the instant of time of the creation of the customer.
- *Start Execution*: the instant of time in which the customer start to be executed from one of the servers.
- *Server ID*: it shows the id of the server if there are more than one.
- *Exit system*: the instant of time in which the customer exits from the system.

NOTE: If the station has a limited queue and there is no empty place, the new customer will be dropped and this event will be logged as *job dropped*.

The panel below the *Simulation Results* shows the status of the servers and the queue.

It has two types of views. The first view shows the utilization of the station. The second view shows the percentage of the current job.

The last pane displays the Simulation time and number of customers. The former displays the speed of the simulation, and the latter displays the total number of the customers. If the simulation is set with limited number of customers, then this area shows the customers remaining. In the bottom there are three buttons

for the simulation Start, Stop and Pause. The user should set the parameters before starting the simulation. However, the parameters can also be changed during the simulation. If the simulation is stopped, the log can be saved as a tab separated text file.

5.6.4 Logger

Logging of events can also be obtained from Markov Chain Simulation. If users wants to see *arrival time, execution time (start and stop times), server id* where the customer is being served for each customer entering the system then they have to select the checkbox in Figure 5.13 where also the link to log file should be provided. The Figure 5.14 shows a sample of log output. The log file can be found in the specified location.

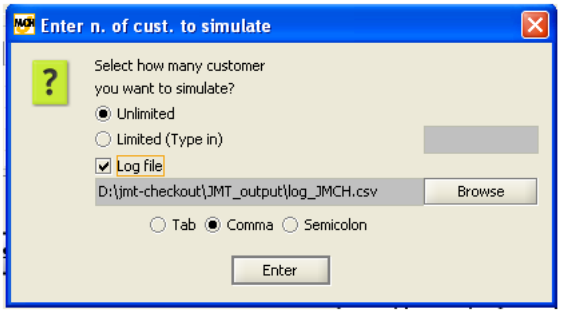


Figure 5.13: Logger file location

Microsoft Excel - log_JMCH.csv

File Edit View Insert Format Tools Data Window Help

A1 Cust. ID

	A	B	C	D	E	F
1	Cust. ID	Arrival Tim	Start Exec	Server ID	Exit System	
2	1	1.39	1.39	0	1.42	
3	2	2.57	2.57	0	2.63	
4	3	2.71	2.71	0	2.79	
5	4	3.31	3.31	0	3.38	
6	5	3.68	3.68	0	3.68	
7	6	4.08	4.08	0	4.15	
8	7	4.83	4.83	0	5.2	
9	8	4.83	5.2	0	5.57	
10	9	6.46	6.46	0	6.58	
11	10	7.99	7.99	0	8.39	

Figure 5.14: Output data: example of a part of a output Log

Chapter 6

JABA (Asymptotic Bound Analysis)

6.1 Overview

Product-form queueing network models are used for modelling the performance of many type of systems, from large computing infrastructures to distributed applications. The complexity of modern systems makes the application of exact solution techniques, such as the convolution algorithm or the MVA, prohibitively expensive in terms of computational resource requirements. Also approximate solution techniques become less accurate or more expensive with the growing complexity of the models. The alternative is represented by *asymptotic* techniques that can efficiently determine asymptotic values for several performance indices such as throughput, response time and queue lengths. The asymptotic techniques are particularly useful in tuning studies where one needs to evaluate the performance gains of different tuning alternatives. The key to determine the asymptotic performances is the knowledge of the queueing center(s) with the highest utilization, i.e. the *bottleneck station(s)*. Multiclass models can exhibit multiple simultaneous bottlenecks depending of the population mix [BS96][BS97]. While identifying the bottleneck stations under a single-class workload is a well-established practice, no simple methodology for multiclass models has yet been found. JABA provides such a technique, called *Polyhedral Analysis* , using convex polytopes based approach presented in [CS04]. An example of application of this technique to identify the bottlenecks in a two-class workload and of the enhancement of the performance of the system is provided in [CS11].

Among the non-bottleneck stations, we distinguish two different types: *dominated* and *masked-off* stations. This distinction is important since the *masked-off* stations, despite having lower utilization than the bottleneck center(s), may still exhibit the largest queue-length and hence the highest response times. *Dominated* stations, instead, typically play a marginal role in determining system performance, and have always the smallest utilization and queue-lengths.

6.1.1 Starting the alphanumeric JABA solver

Selecting button on the starting screen, Figure 6.1 the JABA window shows up.

Three main areas are shown:

Menu : it is organized into three groups of functions. To use a menu, click on the menu heading and choose the appropriate option. For the description of menu entries, see section 6.3

Toolbar : contains some buttons to speed up access to JABA functions (e.g. New model, Open, Save... See section 6.3 for details). If you move the mouse pointer over a button a tooltip will be shown up.

Page Area : this is the core of the window. All JABA parameters are grouped in different tabs. You can modify only a subset of them by selecting the proper tab, as will be shown later.

6.2 Model definition

Models with two or three customer classes provide estimates of wich stations are potential bottleneck. For a brief description of basic terminology please refer to Appendix A.

The workload is characterized for each station by the service demands of every customer class. Al least JABA works with 2 classes of custumers therefore a matrix of service demands is required [LZGS84].

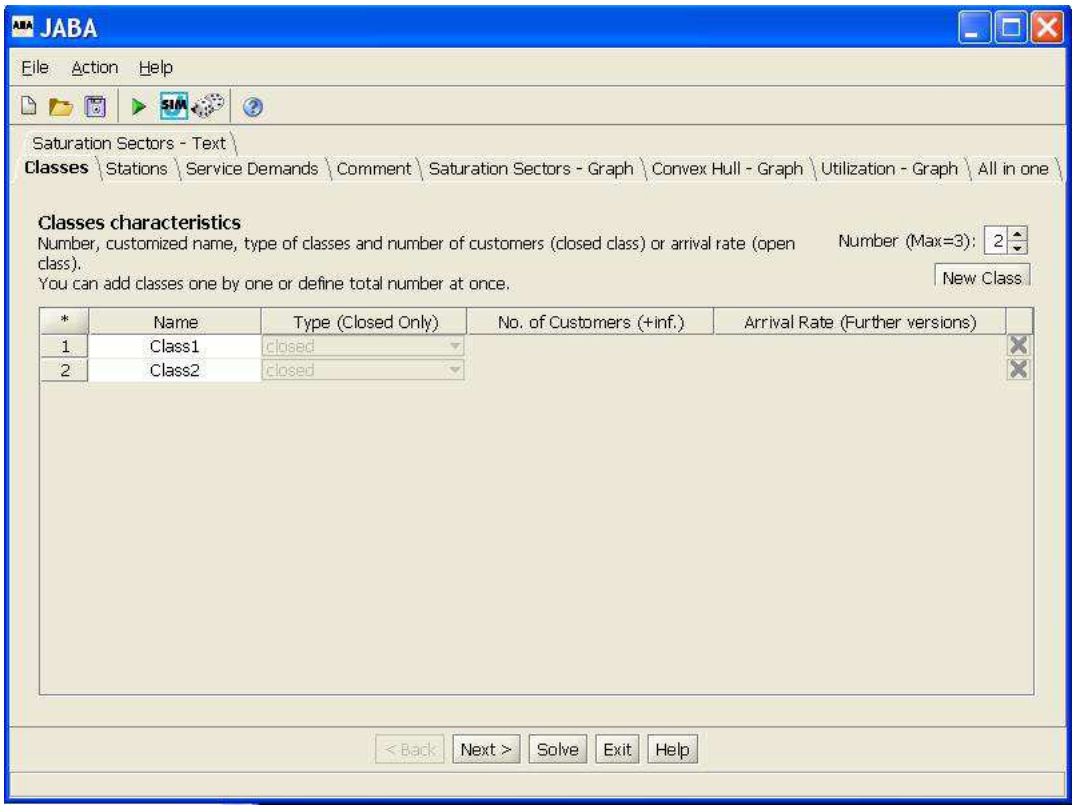


Figure 6.1: Classes tab

6.2.1 Defining a new model

To define a new model select toolbar button or the *New* command from *File* menu. The following parameters must be defined:

1. `Classes`
2. `Stations` (service centers)
3. `Service demands` (or Service Times and Visits)
4. Optional short `Comment`

The execution of JABA produces the following graphs:

- Saturation Sector - graphics
- Convex Hull - graphics
- Utilization - graphics

and a textual report concerning the coordinates of the saturation sectors:

- Saturation Sectors - text

Input tabs

As shown in Figure 6.1, the parameters that must be entered in order to define a new model are divided in four tabs: `Classes`, `Stations`, `Service Demands` and `Comment`.

The number of tabs become five, if you click *Service Times and Visits* button in `Service Demands Tab`. As will be discussed in subsection 6.2.4, the `Service Demands Tab` will be hidden and it will appear `Service Times Tab` and `Visit Tab`. You can navigate across tabs:

- using sequential wizard buttons, if enabled, at the bottom of the window (Figure 6.2)
- using sequential buttons located in menu
- using the tab selector, clicking on the corresponding tab (Figure 6.3)



Figure 6.2: Wizard buttons



Figure 6.3: Tab selector

6.2.2 Classes Tab

A screenshot of this tab can be seen in Figure 6.1. This tab allows to set the number of customer classes. It is possible to define only two or three classes.

The number of classes in the model can be specified in the corresponding input area, shown in Figure 6.4 and can be modified using the keyboard or the spin controls.



Figure 6.4: Number of classes

Using the delete button  associated to a specific class, a class can be removed provided that there will be at least one class after the deletion. Similar result may be obtained using spin controls, decreasing classes number; in this case last inserted class will be removed.

Default class names are *Class1*, *Class2*, ... *ClassN*. The model can be personalized by changing these names.

6.2.3 Stations Tab

The number of stations of the model can be specified in the corresponding input area (Figure 6.5) and can be modified using the keyboard or the spin controls.

Using the delete button  associated to a specific station, a station can be removed provided that there will be at least one station after the deletion. Similar result may be obtained decreasing the number of stations using spin controls; in this case the last inserted station will be removed.

Default station names are *Station1*, *Station2*, ... *StationN*. In order to personalize your model, you can change and give names other than default ones.

In Figure 6.6 there is only one station with default name *Station4* and there are three stations with personalized names: *CPU*, *Disk1* and *Disk2*.

6.2.4 Service Demands, Service Times and Visits Tabs

Service Demands can be defined in two ways:

- directly, by entering Service Demands (D_{kc})
- indirectly, by entering Service Times (S_{kc}) and Visits (V_{kc})

Service demand D_{kc} is the total service requirement, that is the average amount of time that a customer of class c spends in service at station k during a complete interaction with the system, i.e. a complete execution. Service time S_{kc} is the average time spent by a customer of class c at station k for a single visit at that station while V_{kc} is the average number of visits at that resource for each interaction with the system.

Remember that $D_{kc} = V_{kc} * S_{kc}$ so it is simple to compute the service demands matrix starting from service times and visits matrices. Inverse calculation is performed with the following algorithm:

$$V_{kj} = \begin{cases} 1 & \text{if} \quad D_{kc} > 0 \\ 0 & \text{if} \quad D_{kc} = 0 \end{cases}$$

$$S_{kc} = \begin{cases} D_{kc} & \text{if} \quad D_{kc} > 0 \\ 0 & \text{if} \quad D_{kc} = 0 \end{cases}$$

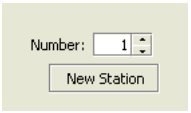


Figure 6.5: Number of stations

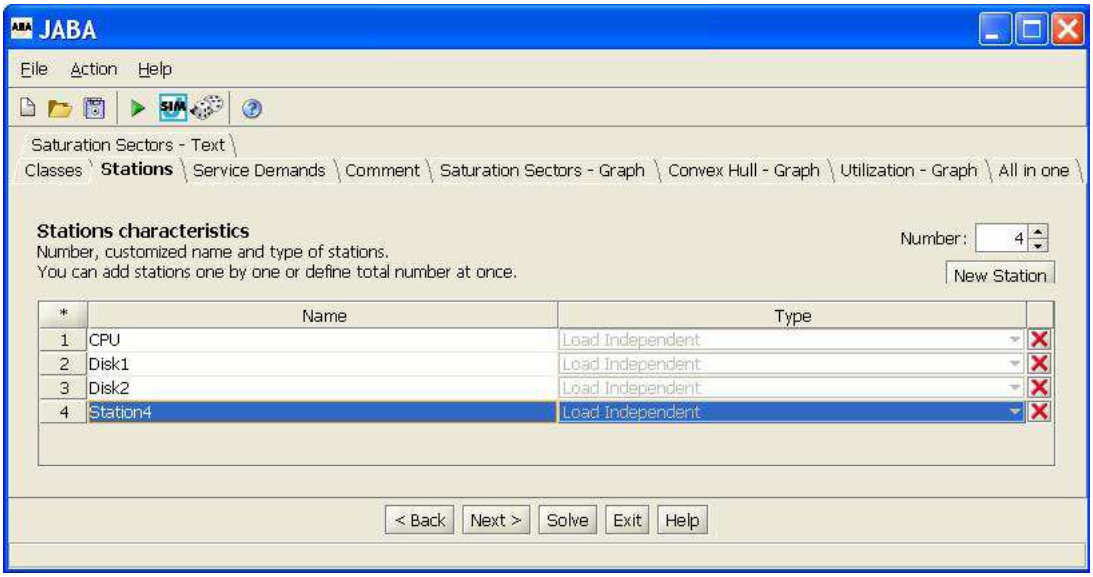


Figure 6.6: Defining the stations name

Service Demands Tab

In this tab you can insert directly Service Demands D_{kc} for each pair {station k-class c} in the model. Figure 6.7 shown a reference screenshot can be seen: notice that a value for every D_{kc} element of the D-matrix has already been specified because the default value assigned to newly created stations is zero.

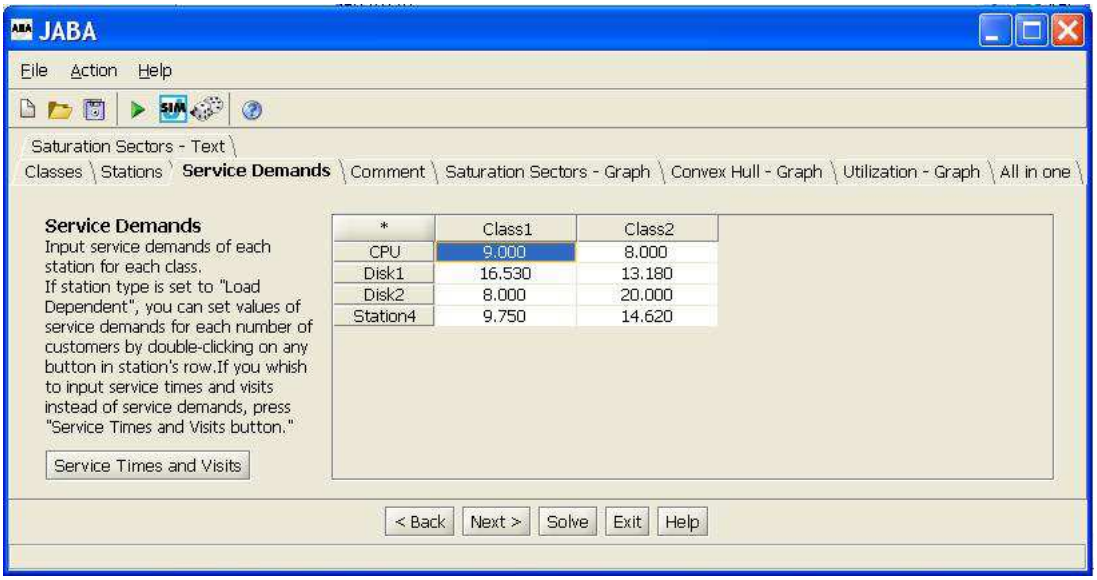


Figure 6.7: The Service Demands Tab

In Figure 6.7, each job of *Class1* requires an average service demand time of 6 sec to *CPU*, 16.53 sec to *Disk1*, 8 sec to *Disk2* and 9.75 sec to *Station4*.

Service Times and Visits Tabs

In the former tab you can insert the Service Times S_{kc} for each pair {station k-class c} in the model, in the latter you can enter the visits number V_{kc} (See Figure 6.8).

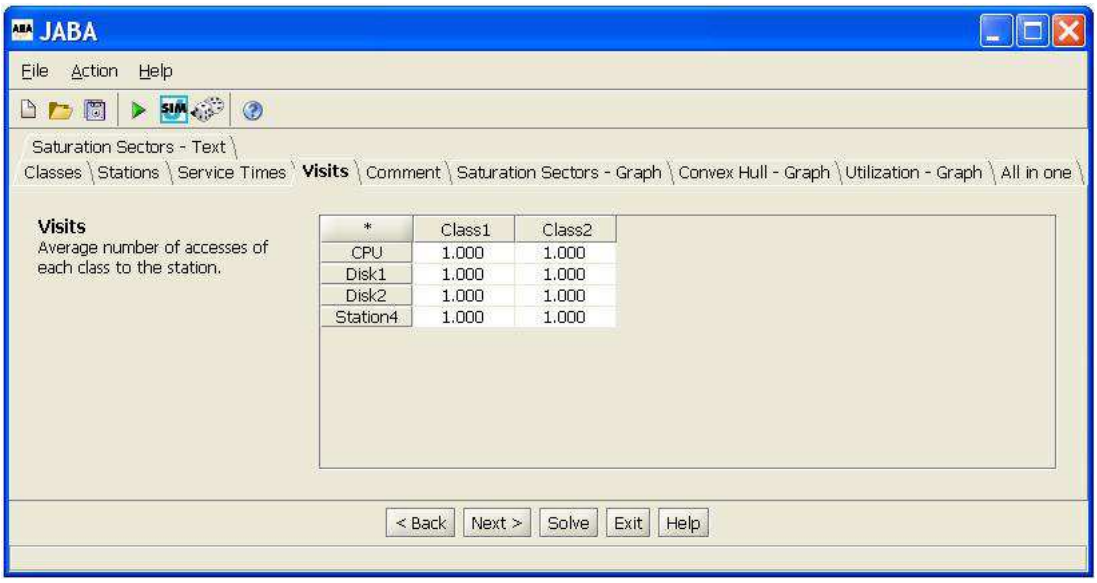


Figure 6.8: Visits Tab

The layout of these tabs is similar to the one of the `Service Demand Tab` described in the previous paragraph. The default value for each element of the Service Times (S) matrix is zero, while it's one for Visits' matrix elements.

6.2.5 Comment Tab

In this Tab, a short - optional - comment about the model can be inserted; it will be saved with the other model parameters.

6.2.6 Saturation Sector - Graph

Using the `Solve` command the Saturation Sector graph appears.

Two-class workload

The population of the system (i.e., the number of customers of the different classes in execution) is defined with the vector $\mathbf{N}=(N_1, N_2)$ where N_1 is the number of customers belonging to class1 and N_2 is the number of customers of class2. We define the *population mix* corresponding to the population $\mathbf{N}$ as the vector $\boldsymbol{\beta} = \{\beta_1, \beta_2\}$ whose components are all non-negative and are given by

$$\beta_r = \frac{N_r}{N} \qquad \sum_r \beta_r = 1 \tag{6.1}$$

We study the asymptotic behavior of a system under a particular type of population growth that consists of letting the total number of customers N in the system to grow to infinity keeping constant the population mix $\boldsymbol{\beta}$ and that we call proportional population growth. In multiclass queueing networks the ratio between the (global) utilizations of two arbitrary service stations is not constant, but depends on the population mix. Thus, in general, different population mixes can yield different bottlenecks. Under this assumption of population growth, we define as *system bottlenecks* the stations for which the following expression holds:

$$\lim_{N\to\infty} U_i(\mathbf{N}) = \lim_{N\to\infty} \sum_r U_{ir}(\mathbf{N}) = 1. \tag{6.2}$$

These stations will be referred also as *saturating stations*. In general, different values of $\boldsymbol{\beta}$ may yield different sets of system bottlenecks. Special types of system bottlenecks are the *natural bottlenecks* . The class-r natural bottleneck is the station that saturates when the workload consists only of class-r customers ($\beta_r = 1$) and the number of customers grows to infinity.
In the more general case in which different classes have distinct natural bottlenecks, proportional population growths corresponding to different population mixes may identify different sets of bottleneck stations. This means that, keeping the total population N of the system constant (and large) and varying the population

mix, we may observe the *bottleneck migration* phenomenon. In this situation, it is possible to find a set of mixes, represented as an interval $[\beta_1', \beta_1'']$, where multiple stations are bottlenecks, i.e., saturate concurrently. We define this interval as *saturation sector* and its edges correspond to *switching points* where stations may change their bottleneck/non-bottleneck status. Depending on the values of the service demands a system may have one or more saturation sectors.

JABA represents the saturation sectors and the switching points through a two-dimensional graph, see e.g. Figure 6.9.

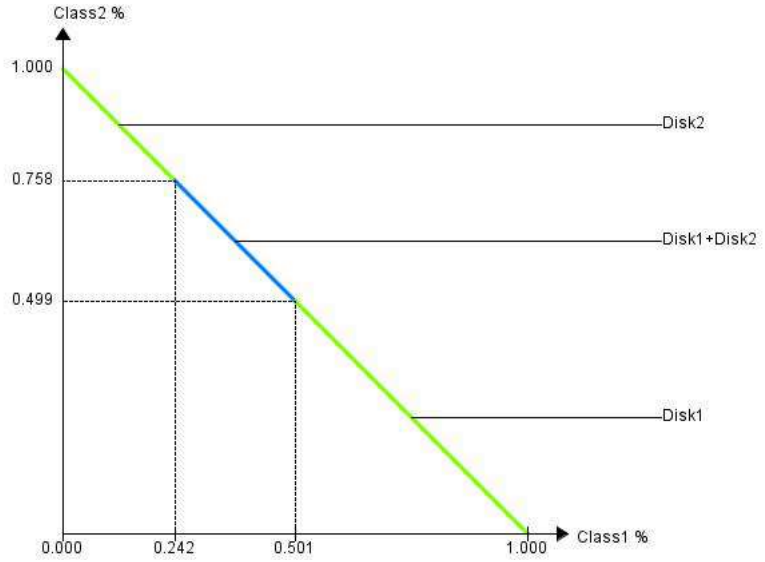


Figure 6.9: Saturation Sector graph. The labels refer to the stations that saturate in the corresponding interval.

In the example of Fig.6.9 we may observe that *Disk1* is a bottleneck for the mixes population from $\beta_1' = 0.242$ to $\beta_1'' = 1$. The station *Disk2* is a bottleneck for the mixes population from $\beta_1' = 0$ to $\beta_1'' = 0.501$. The switching points are (0.242, 0.758) and (0.501, 0.499). The blue sector represents the mixes where two stations are bottlenecks concurrently. Green sectors represent the mixes where only one station saturates. In the case where more than two stations saturate concurrently, red color will be used.

Three-class workload

Sometimes a two-class model does not describe properly the workload you want to study, because more than two clusters of customers can be identified. In this case JABA allows you to define a three-class model to fit better the behavior of the customers of the system. Exactly for the two-class, also for the three-class model the stations for which holds Equation 6.2 are called *system bottlenecks* and will be referred also as *saturating stations*. In this context $\boldsymbol{\beta}$ is a three-dimensional vector and different values of $\boldsymbol{\beta}$ may lead to different sets of *system bottlenecks*. JABA uses a triangle to show for each value of $\boldsymbol{\beta}$ which stations will be bottlenecks. The vertices of this triangle represent the values [1,0,0], [0,1,0] and [0,0,1] of $\boldsymbol{\beta}$. The inside of this triangle will be divided into sub-areas and each area represents a population mix range where one, two or more stations are bottleneck at the same time. It is clearly understandable that in this situation there are no no more *switching points* (such as in the two-class model) but instead *switching segments* . In the example of Figure 6.10 we can observe a big area,on the top of the triangle, that rapresents the range of mix population where only *Station4* is a bottleneck. In the center of the triangle there is a little red area that represent the range of mix population where *Disk1*, *Disk2* and *Station4* are all bottleneck. The color of a sub-area has the following meaning:

- just one station is bottleneck for the values of $\boldsymbol{\beta}$ identified by a *yellow* area.
- two stations are bottleneck for the values of $\boldsymbol{\beta}$ identified by a *brown* area.
- more than two stations are bottleneck for the values of $\boldsymbol{\beta}$ identified by a *red* area.

This graph is useful for understanding when stations became bottlenecks; sometimes is not simple to get the exact values of the *switching segment*, in order to obtain them select the "Saturation Sector - Text" tab subsection 6.2.10.

6.2.7 Convex Hull - Graph

Using the `Solve` command and selecting the "Convex Hull - Graph" tab the Convex Hull graph will be shown.

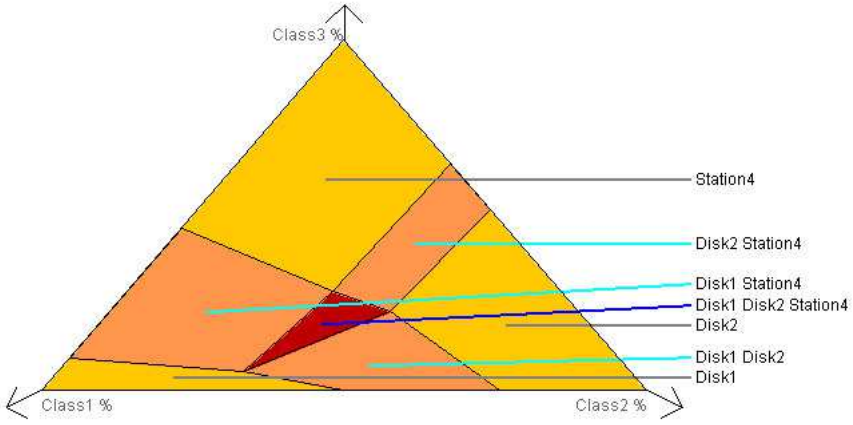


Figure 6.10: Saturation Sector graph

The Convex Hull graph implements the *polyhedral analysis* technique, a technique, based on the theory of convex polyhedra, for identifying the bottlenecks of multi-class queueing networks with constant-rate servers.

Polyhedral technique

Let $\mathbf{R} = \{1, 2, \ldots, r\}$ be the set of customer classes indices and $\mathbf{M} = \{1, 2, \ldots, m\}$ the set of station indices. The average service demand required by class-r customers on station i will be indicated with $D_{i,r}$. The following matrix describes compactly the system

$$\mathbf{D} = \begin{bmatrix} D_{1,1} & D_{1,2} & \cdots & D_{1,r} \\ D_{2,1} & D_{2,2} & \cdots & D_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ D_{m,1} & D_{m,2} & \cdots & D_{m,r} \end{bmatrix}$$

This technique introduces a taxonomy of stations. *Potential bottleneck* set $\mathbf{\Phi}$ is the set of stations that can become bottlenecks for some feasible population of the system. The stations that cannot become bottlenecks for any feasible population are grouped in the set of non-bottleneck stations $\mathbf{\Psi} = \mathbf{M} \setminus \mathbf{\Phi}$. Among these it is possible to distinguish two kinds of stations: a station $j \in \mathbf{\Psi}$ is said to be *dominated* by $i \in \mathbf{M}$ iff $D_{jr} < D_{ir}, \forall r \in \mathbf{R}$ and all non-dominated stations $k \in \mathbf{\Psi}$ are called *masked-off* stations. It is known [CS04] that all potential bottlenecks lie on the boundary of the *characteristic polytope* of the system . Viewing the matrix $\mathbf{D}$ as a set of m points in a r-dimensional space, the *characteristic polytope* of a system is the convex-hull of the set of points $\mathbf{D} \cup proj(\mathbf{D})$, where $proj(\mathbf{D})$ is the set of all vectors derived by a by setting to zero k of its components, for all possible $k = 1, \ldots, r$ and for all possible positions.

JABA shows which stations of a system may saturate building the *characteristic polytope*.

Two-class workload

In two-class models each station corresponds to a point on a graph whose co-ordinates are represented by the service demand of the two customer classes. For example, a station with service demands 10 sec for class 1 and 15 sec for class 2 is represented as a point with coordinates (10,15).

In the example of Figure 6.11, we can see that *Disk1* and *Disk2* (depicted in red) lie on the boundary of the convex hull, thus are potential bottlenecks. *CPU* (green) is instead a *dominated* non-bottleneck station, since it is dominated by *Station4* and *Disk1*. *Station4* (blue) is also in the interior of the convex hull, but it is not *dominated*, so it is a *masked-off* station.

In order to gain exact information about the coordinates of a point it is sufficient to move the mouse on it and additional information will appear after the station name. Clicking (left button) on the line that it connects two bottleneck stations you can obtain information about the saturation sector such as the interval of mix population that make possible the bottleneck and also the throughput of the two classes.

Now suppose that we have to analyze the upgrade of the station *Station4*, the new *Station4* will have half demand for both the customer's classes compared to the old one. In order to do this we can change the value in the demand's table of JABA or click on the point (left button) and drag it in the new position. While dragging the point temporary co-ordinates are shown and when the point is released all data are update and a new graph is painted. As showed in the Figure 6.13, it is possible that a area of the graph could be crowded of points and

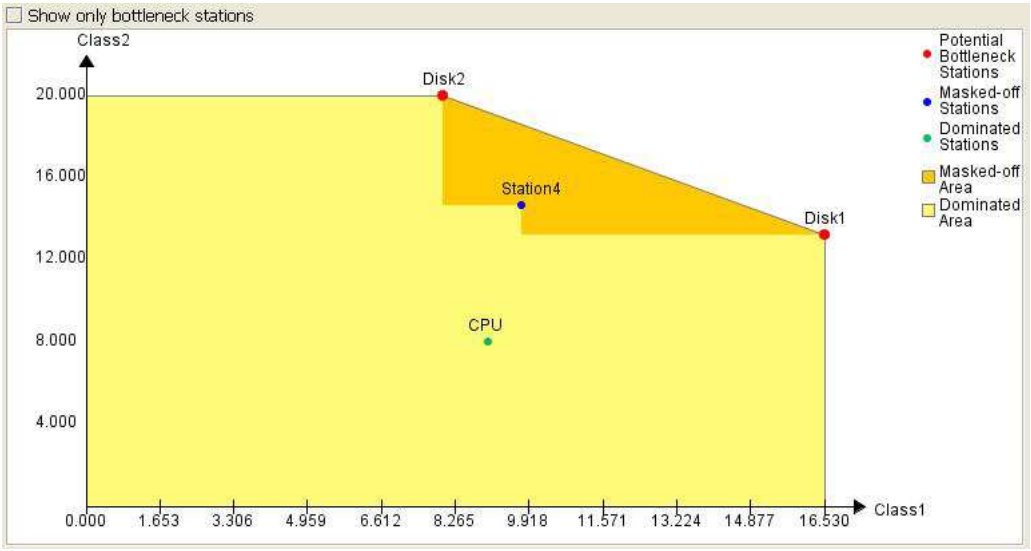


Figure 6.11: Convex Hull graph

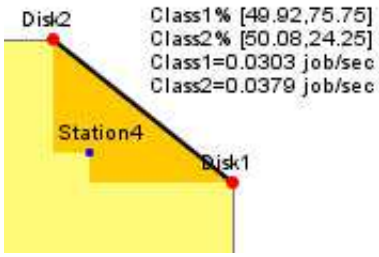


Figure 6.12: Convex Hull graph while saturation sector is selected

it is difficult to read the name of the station; at first we can try to change the zoom, we can therefore zooming in or out with a right-click on the graph.

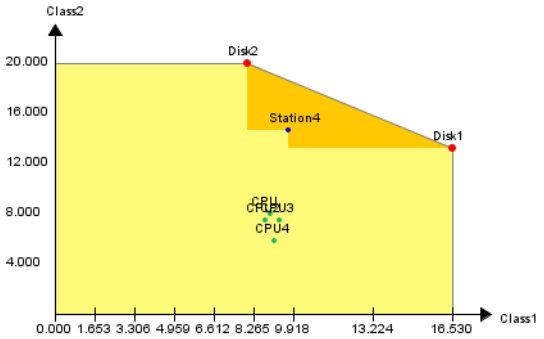


Figure 6.13: Convex Hull graph with a crowded area

If the zooming is no enough we can try filtering the zone and freeing only some points. In order to filter the area is enough to select it keeping the mouse's left button pressed. The background of the selected area will become light-blue and the labels of points will disappear. Now we try to reduce the filtered area adding some free area; to do that we select the area that we wish to free keeping the mouse's right button pressed. In the following example we have set the tallest point free. Only the name of the station that is in the free area is showed.

Three-class workload

If you define a three-class model the convex-hull is a 3d object (Figure 6.15), dragging this three-dimensional object JABA allows you to rotate it showing its different faces. Beside the graph JABA will show a set of options that allow you to

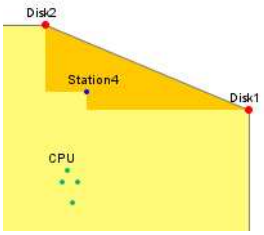


Figure 6.14: Convex Hull graph with a filtered area

- set the render mode (solid, wireframe, points)
- hide or show the names of the stations
- hide or show the non-bottleneck stations
- select a station

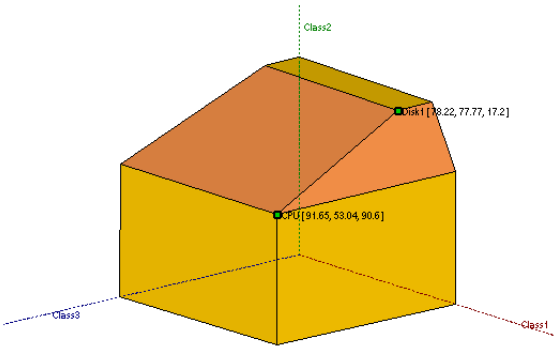


Figure 6.15: Convex Hull graph with a 3 class workload

6.2.8 Utilization - Graph

If you are working on a two-class model using the `Solve` command and selecting the "Utilization - Graph" tab the Utilization graph will be shown. Using a graphical method JABA shows you the behavior of the utilization of the stations varying the population mix of the system.

Graphical method

This method takes as input the description of the system and produces a graph showing the behavior of the utilization for every stations.

Considering a system with two class of customers and m stations the following matrix describes compactly the system

$$\mathbf{D} = \begin{bmatrix} D_{1,1} & D_{1,2} \\ \cdots & \cdots \\ D_{m,1} & D_{m,2} \end{bmatrix}$$

Where $D_{i,j}$ is the average service demand required by class-j customers on station i.

In order to explain this method we will use a running example with the matrix

$$\mathbf{D} = \begin{bmatrix} 9 & 8 \\ 16.530 & 13.180 \\ 8 & 20 \\ 9.750 & 14.620 \end{bmatrix}$$

The output of the method will be a graph of m functions. Every function is related with a station, it shows the value of the utilization of the related station for every possible population mix of customers. In order to plot the functions will be used a Cartesian-plane, on the abscissas the percentage of class-1 customers and on the ordinates the utilization Figure 6.16.

The following proposition describes the behavior of the utilization function and it is very useful for our goal. ***Proposition 1:*** the utilization of every stations is a *polygonal chain* and slope changes may occur only when the abscissa is equal to the first component of a *switching-points* (the output of *Saturation Sector Graph*).



Figure 6.16: The Cartesian plane used to plot the Utilization-Graph.

In order to highlight this behavior we divide the abscissa-axis into segments using the *switching-points*, from now on we will indicate with $\mathbf{S} = \{1, \ldots, s\}$ the set of this segments. In Figure 6.17 it is shown this division for our running example.

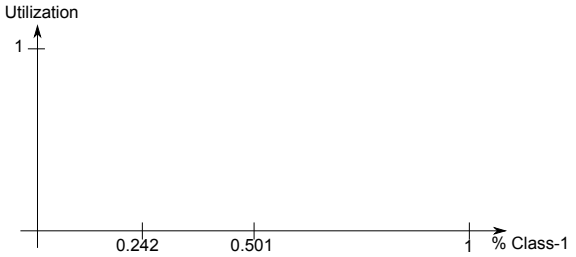


Figure 6.17: Division of the abscissa-axis of the Utilization-Graph.

Thanks to *Proposition 1* all we need is

- for every segment s in $\mathbf{S}$ the slope of the utilization of every station i, indicated as $slope_{s,i}$
- for every station i the value of the utilization when the abscissa is equal to zero, indicated as $U_i(0)$

Indicating with b_s the station that saturates in the segment $s \in S$ [1], the following formulas give us these information

$$slope_{s,i} = \begin{cases} 0 & \text{if s is a saturation sector} \\ \frac{D_{i,1}}{D_{b_s,1}} - \frac{D_{i,2}}{D_{b_s,2}} & \text{otherwise} \end{cases} \tag{6.3}$$

$$U_i(0) = \frac{D_{i,2}}{D_{b_1,2}} \tag{6.4}$$

Using Equation 6.3 and Equation 6.4 JABA plots for every station the behavior of the utilization, Figure 6.18.

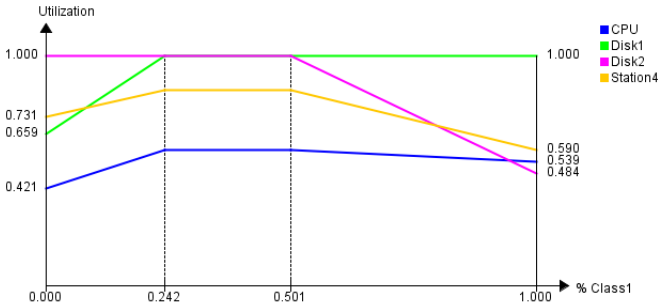


Figure 6.18: Utilization graph

6.2.9 All In One - graph

If you are working on a two-class model using the `Solve` command and selecting the `All In One` tab the Saturation-Sector graph, the Convex-Hull graph and the Utilization graph will be shown together. If you use this type of composite graph, it is interesting to point out that you may change with the mouse a point in the Convex Hull (i.e., a value of the service demand of a station) and immediately you will see the effects on the other two graphs (i.e., Saturation Sector and Utilization graphs) since they are updated online.

[1] This is meaningful only for non saturation sectors.

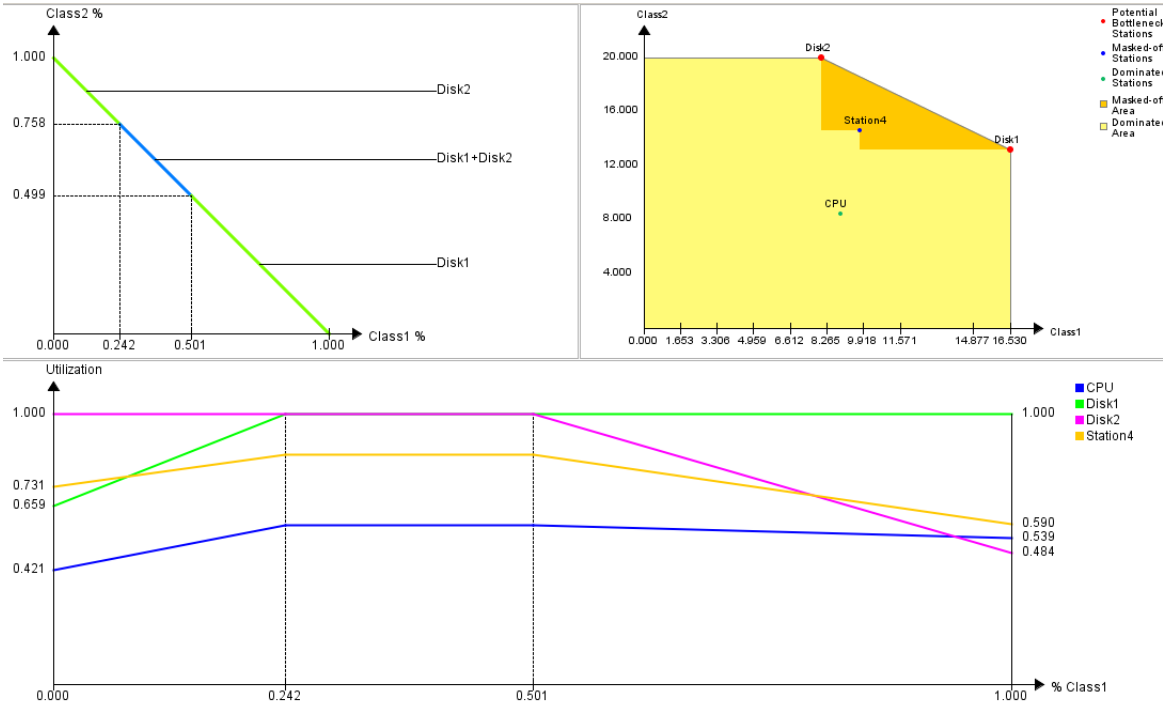


Figure 6.19: *All In One* graph

6.2.10 Saturation Sector - Text

It's a simple report about the saturation sector, for each saturation sector you can find:

- The station that are bottleneck
- The switching point or the switching segment bound points.

This tab could be helpful in the Saturation Sector graph of three classes of costumers.subsection 6.2.6

6.2.11 Modification of a model

To modify system parameters return to the main window and enter new data or moving a station in Convex Hull - graph tab (see subsection 6.2.7). After the modifications, if you use `Solve` command, the new graphs will show. You can `save` this new model with the previous name - overwriting the previous one - or `save` it with a different name or in a different directory.

6.3 Menu entries

6.3.1 File

New

Use this command in order to create a new JABA model.
Shortcut on Toolbar: 
Accelerator Key: CTRL+N

Open

Use this command to open an existing model. You can only open one model at time, to edit two or more models start more than one instance of JABA. If current model was modified since its creation or last save action, a popup window will be shown for confirmation.

It's possible to open not only models saved with JABA (*.jaba), but also with other programs of the suite (for example JMVA *.jmva, JSIM *.jsim and JMODEL *.jmodel). Whenever a foreign data file is opened, a conversion is performed and error/warnings occurred during conversion will be reported in a window.

Models are stored in XML format, see *JMT system manual* for a detailed description.

Shortcut on Toolbar: 
Accelerator Key: CTRL+O

Save

Use this command in order to save the active document with its current name in the selected directory.

When you save a document for the first time, JABA displays the Save As dialog box so you can rename your document. If you save a model after its resolution, results are stored with model definition data.

Shortcut on Toolbar: 
Accelerator Key: CTRL+S

Exit

Use this command in order to end a JABA session. You can also use the Close command on the application Control menu. If current model was modified since its creation or last save action, a popup window will be shown for confirmation.

Accelerator Key: CTRL+Q

6.3.2 Action

Solve

Use this command when model description is terminated and you want to start the solution of the model. At the end of the process the Saturation Sector - graph subsection 6.2.6 will show.

Shortcut on Toolbar: 
Accelerator Key: CTRL+L

Randomize

Use this command in order to insert random values into Service Demands - or Service Times - table.

Shortcut on Toolbar: 
Accelerator Key: CTRL+R

6.3.3 Help

JABA Help

Not still avaiable.

6.4 Examples

In this section we will describe some examples of model parametrization and solution using JABA exact solver. Step-by-step instructions are provided in two examples:

1. A two class model (subsection 6.4.1)
2. A three class model (subsection 6.4.2)

6.4.1 Example 1 - A two class model

Solve the two class model specified in Figure 6.20.

There are three stations (named *CPU*, *Disk1* and *Disk2*). Service demands for each station are reported in Table 6.1.

	CPU	Disk1	Disk2
FirstClass	2.60	7.02	4.81
SecondClass	7.02	4.51	5.70

Table 6.1: Example 1 - service demand

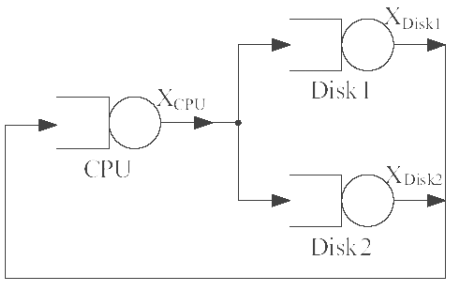


Figure 6.20: Example 1 - network topology

Step 1 - Classes Tab

- use New command to create a new JABA document
- by default, you have already two classes
- if you like, substitute default *Class1* and *Class2* name with a customized one (*FirstClass* and *SecondClass* in our example)

At the end of this step, the `Classes Tab` should look like Figure 6.21.

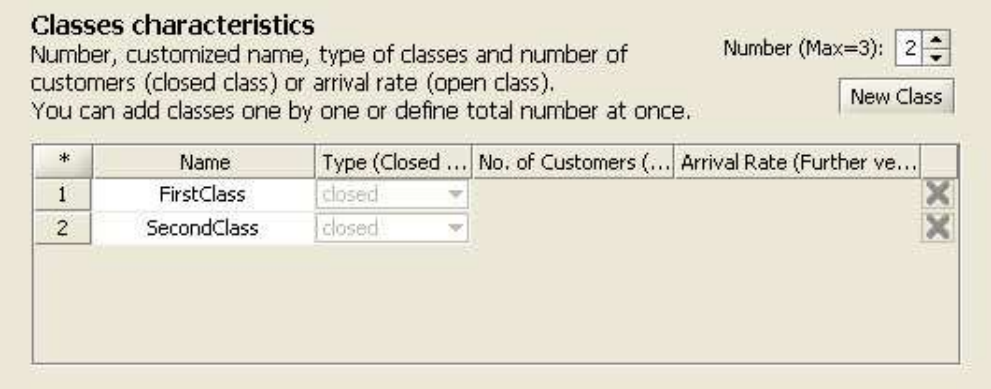


Figure 6.21: Example 1 - input data (Classes Tab)

Step 2 - Stations Tab

- use `Next >` command to switch to `Stations Tab`
- digit number 3 into stations number textbox or select number 3 using spin controls or push *New Station* button twice. Now your model has three stations with a default name
- if you want you can change station names. Substitute *CPU* for default name *Station1*, substitute *Disk1* for default name *Station2*, substitute *Disk2* for default name *Station3*

At the end of this step, the `Stations Tab` should look like Figure 6.22.

Step 3 - Service Demand Tabs

- use `Next >` command to switch to `Service Demands Tab`
- you can input all Demands in the table.
- press *Service Time and Visit* button if you don't know the Service Demands of the three stations. After button pressure, the `Service Demands Tab` will be hidden and `Service times Tab` and `Visit Tab` will appear. Now Service Times and number of Visits should be typed.

At this point, the `Service Demands Tab` should look like Figure 6.23.

Step 4 - Saturation Sectors - Graphics

Use `Solve` command to start the solution of input model. In the *Saturation Sector - Grlaphic Tab* will be displayed a graph like the one of Figure 6.24.

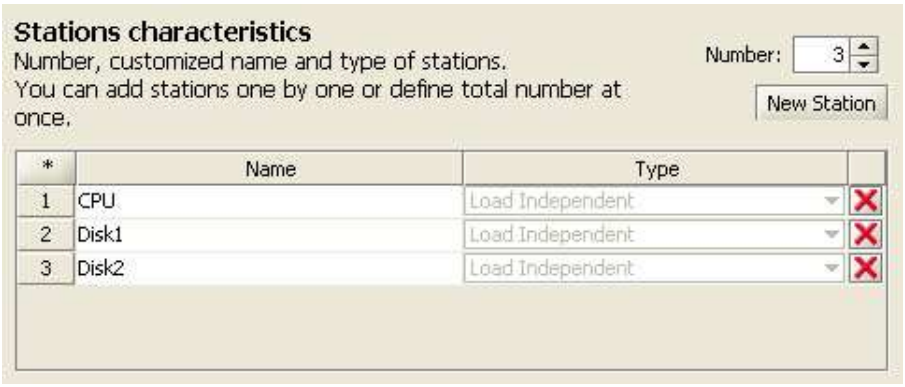


Figure 6.22: Example 1 - input data (Stations Tab)

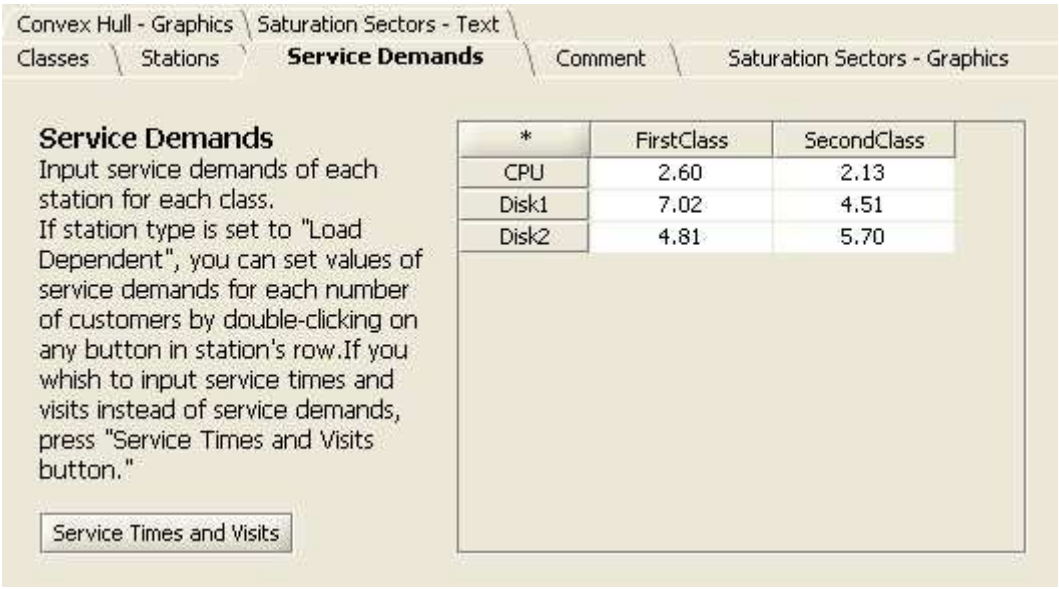


Figure 6.23: Example 1 - input data (Service Demand Tab)

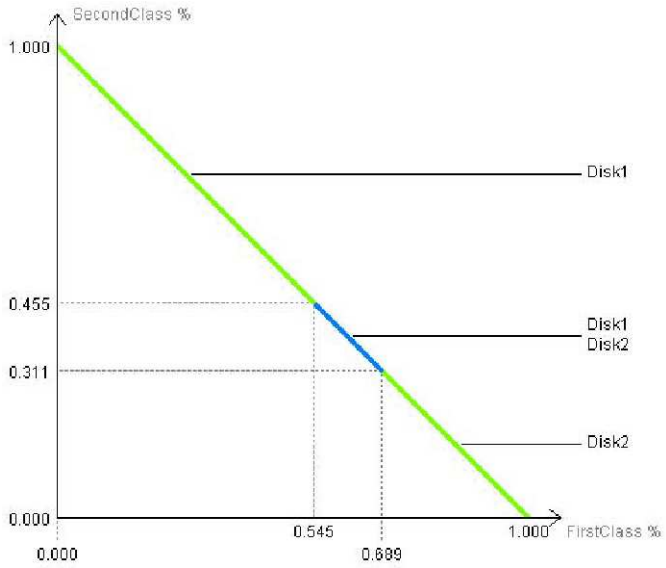


Figure 6.24: Example 1 - Saturation Sectors Graphics

This graph show the range of mix population where two or more stations are both bottleneck. In this example *Disk1* and *Disk2* are both bottleneck when *FirstClass* is among *0.545 - 0.689* and *SecondClass* is among *0.311 - 0.455*

Step 5 - Convex Hull - Graphics

Using `Next >` command to switch to `Convex Hull - Graphics` and a graph like Figure 6.25 will be shown.

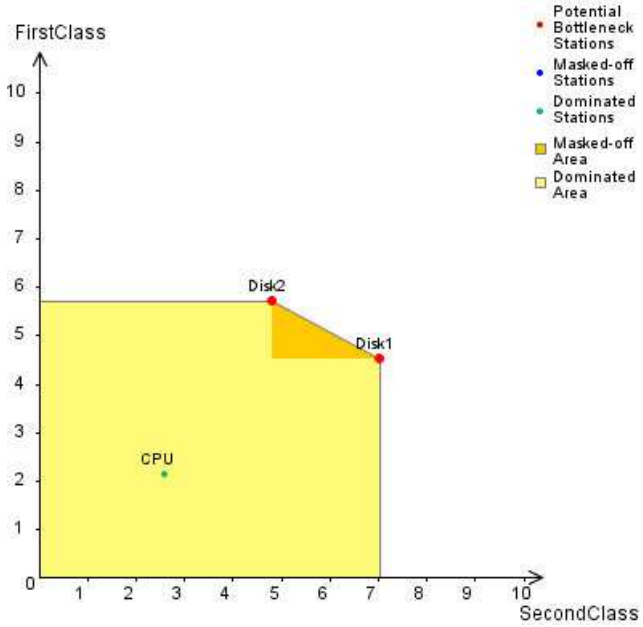


Figure 6.25: Example 1 - Convex Hull - Graphics

This graph implements the polyhedra analysis technique. The points on this graph represent the stations. Dominated stations are green, masked-off ones are blue and bottlenek ones are red. If you click on a station, additional information will be shown, otherwise you can drag a station and see what will change in your model.

Step 6 - Saturation Sectors - Text

If you use `Next >` command again you will see the *Saturation Sectors - Text*, a simple textual report about the Saturation Sectors.

6.4.2 Example 2 - A three class model

Solve the multiclass open model specified in Figure 6.26. The model is characterized by three A , B and C.

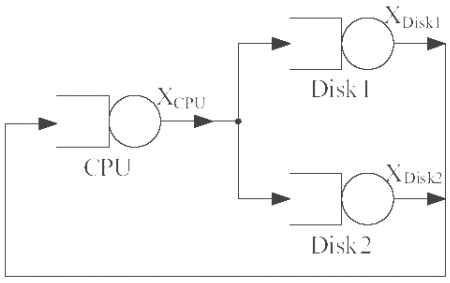


Figure 6.26: Example 2 - network topology

There are three stations, identified with names *CPU*, *Disk1* and *Disk2*. Service times and visits for stations are shown in Table 6.2 and Table 6.3.

Step 1 - Classes Tab

It is the same procedure of the Example 1 (subsection 6.4.1). You have only to add one more class.

	CPU	Disk1	Disk2
Class A [s]	0.01	0.38	0.30
Class B [s]	0.02	0.62	0.80
Class C [s]	0.09	0.42	0.50

Table 6.2: Example 2 - service times

	CPU	Disk1	Disk2
Class A	101.0	60.0	40.0
Class B	44.0	16.0	27.0
Class C	63.0	21.0	35.0

Table 6.3: Example 2 - number of visits

Step 2 - Stations Tab

It is the same procedure of the Example 1 (subsection 6.4.1).

Step 3 - Service Times and Visits Tabs

- use `Next` > command to switch to `Service Demands Tab`
- press *Service Time and Visit* button if you don't know the Service Demands of the three stations, Now Service Times and number of Visits should be typed. After button pressure, the `Service Demands Tab` will be hidden and `Service times Tab` and `Visit Tab` will appear
- you can input all Service Times in the table.

At this point, the `Service Times Tab` should look like Figure 6.27.

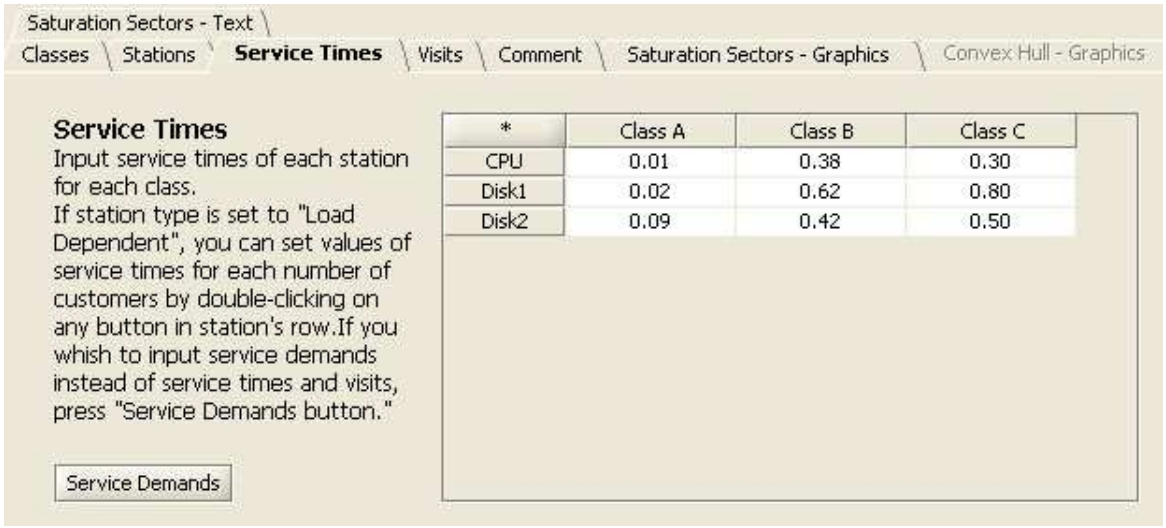


Figure 6.27: Example 2 - input data (Service Times Tab)

- use `Next` > command to switch to `Visits Tab`
- input number of visits for each center in the table.

At the end of this step, the `Visits Tab` looks like Figure 6.28.

Step 4 - Saturation Sectors - Graphics

Use `Solve` command to start the solution of the input model. In the *Saturation Sector - graph Tab* will be displayed a graph like the one of Figure 6.29.

This graph show the range of mix population where two or more stations are both bottleneck. For example the orange area is a renge of mix population in which *Disk1* and *Disk2* are both bottlenek.

Step 5 - Saturation Sectors - Text

If you use `Next` > command again you will see the *Saturation Sectors - Text*, a simple textual report about the Saturation Sectors.

Visits
Average number of accesses of each class to the station.

*	A	B	C
CPU	101.00	60.00	40.00
Disk1	44.00	16.00	27.00
Disk2	63.00	21.00	35.00

Figure 6.28: Example 2 - input data (Visits Tab)

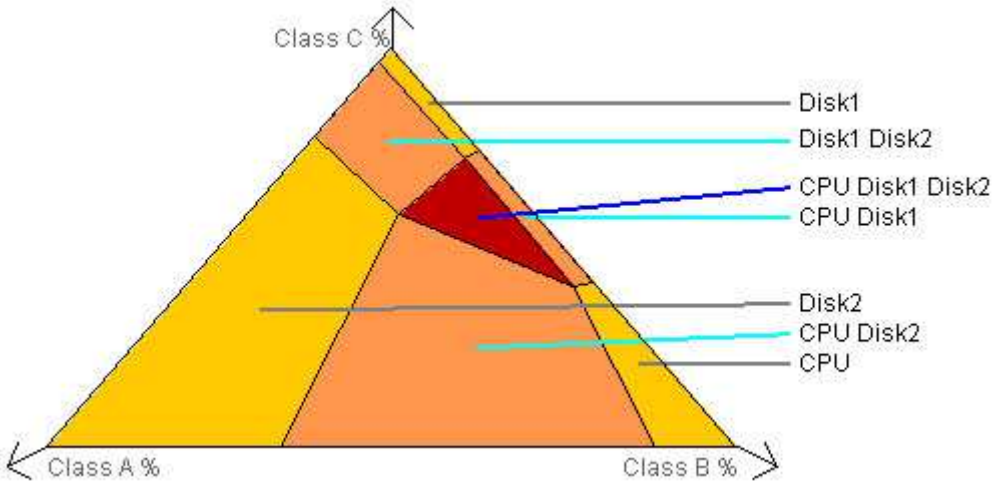


Figure 6.29: Example 2 - Saturation Sectors Graphics

Chapter 7

JWAT - Workload Analyzer Tool

A computer system can be viewed as a set of hardware and software resources that are used in a time-variant fashion by the processing requests generated by the user. During any given time interval, the users submit their processing requests to the systems by inputting sets of programs, data, commands, requests for web sites or file downloads, etc. All these input requests are usually designated by the collective term of *workload*. Every time the value of a system or network performance index is given, the workload under which that values obtained must be reported or, at any rate, known, since performance cannot be expressed by quantities independent of the system's workload. Thus, no evaluation problem can be adequately solved if the workloads to be processed by the systems or transmitted over a network are not specified.
The quantitative description of a workload's characteristics is commonly called *workload characterization*. A characterization is usually done in terms of workload parameters that can affect system behavior and are defined in a form suitable for a desired use. Thus, a workload is characterized correctly if, and only if, its result is a set of quantitative parameters selected according to the goals of the characterization operation. Workload characterization is fundamentally important in all performance evaluation studies and is indispensable for designing useful workload models.
A workload may be regarded as a set of vectors in a space with a number of dimensions equal to the number of the workload parameters used to describe each element considered. Since the amount of resulting data is generally considerable, their analysis for the purpose of identifying groups of components with similar characteristics will have to be performed by statistical techniques that can handle multidimensional samples, that is, techniques that deal with multiple factors.

The JWAT tool is based on the application of a set of statistical techniques oriented towards the characterization of log data representing resource utilizations, traffic of requests, user web paths, etc. Its application provides the input data for the models to be used in all types of performance evaluation studies. Functions for *fitting* experimental data in order to obtain the values that can be used to parameterize the arrival rates and service times of the model are provided.

The main window of JWAT (Figure 7.1) is opened pressing the 

button in the JMT suite main window. Through this window, it is possible to select one of the three main components of JWAT:


Workload Analysis: Characterization of the workload of a system through the analysis of the log files applying a clustering algorithm (k-Means or Fuzzy k-Means) and other statistical techniques. The input data can be loaded directly from standard log files (e.g., Apache log files, IIS) or from files with any format described by the users. Several graphical representations of the results are also provided for their statistical comparison (histograms, pie-charts, dispersion matrix, QQ-Plot, QQ-Plot matrix, scatter plots, ...).


Fitting: Application of statistical hypothesis testing on the data of a log file for both Pareto and Exponential distribution and subsequent estimation of the parameters. Several graphic representations are provided in order to make a visual assessment of the fitting of the data to the available distributions: QQ-Plot, cumulative distribution function (CDF) and probability density function (pdf).


Traffic Analysis: Characterization of the arrival traffic of requests analyzing the timestamp in a log file and deriving the parameters that refer to the burstiness of requests

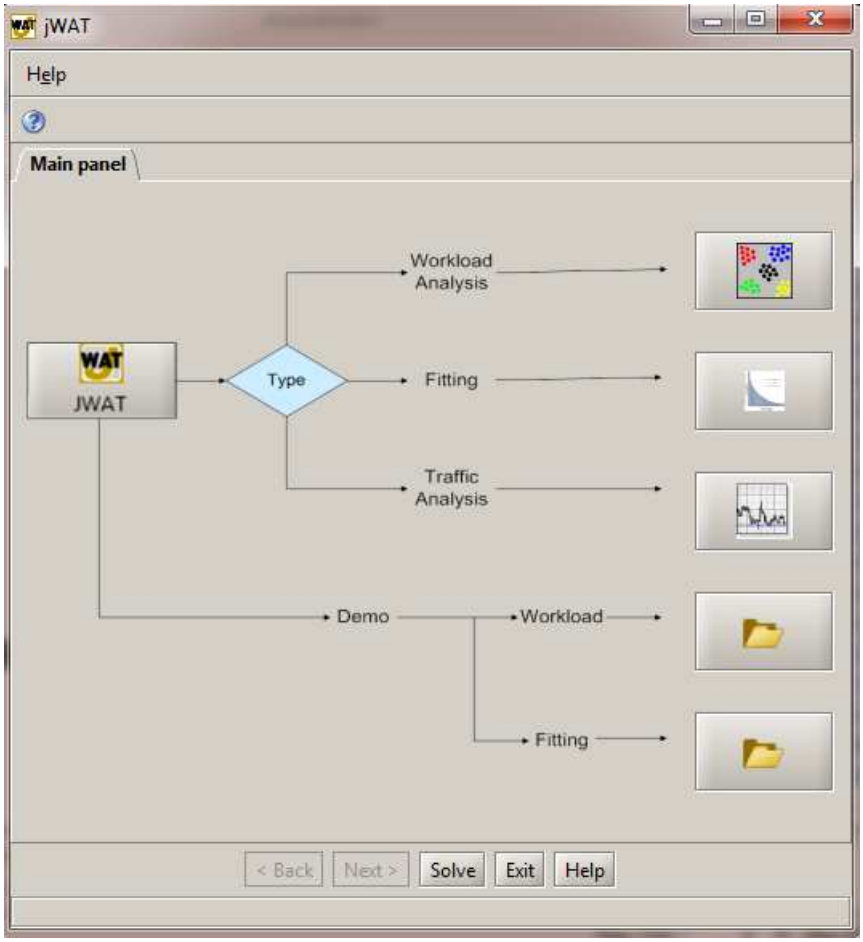


Figure 7.1: Main window of JWAT - Workload Analyzer Tool

7.1 Workload analysis

To perform a workload analysis press the button  of the JWAT window and the window of Figure 7.2 will be opened. This application help the user to perform in the proper sequence the various steps of the

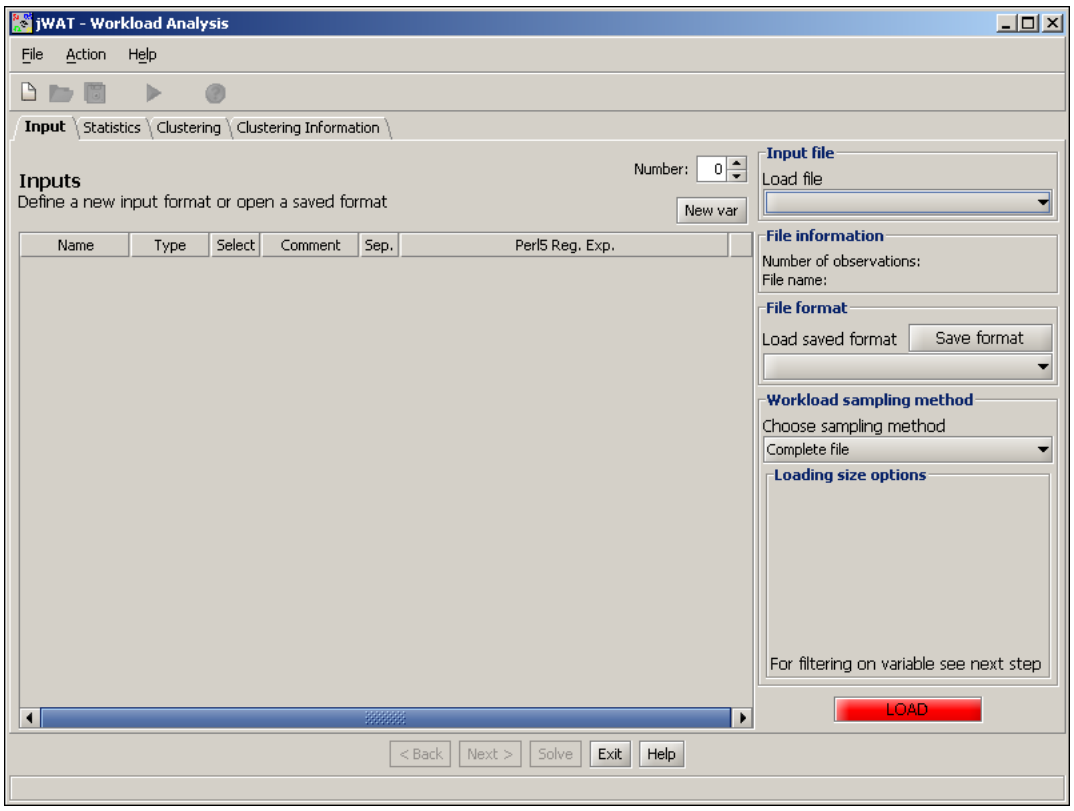


Figure 7.2: Main window for the Workload analysis

workload characterization from the data loading to the statistical analysis, to the results visualization and to their evaluation. The main components of the workload analysis window are:

- *Menu*: three groups of functions are available: File, Action, Help. To utilize the menu select the desired option. For the description of menu options see Section subsection 7.1.8.
- *Toolbar*: several buttons are available to facilitate and to speed up the access to the functions (e.g, New input file, Help ... see Section subsection 7.1.8). When the cursor is over a button tooltips appear.

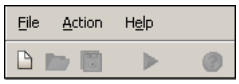


Figure 7.3: Application toolbar

- *Tabbed pane*: It concerns the main functions of the application. All the operations that can be performed on the input data are shown in four navigable tabs: input, statistics, clustering, clustering information (Figure 7.4).



Figure 7.4: Tabs selector

- *Navigation buttons*: used for navigate between panels. The buttons are enabled and disabled automatically depending on the operations to be performed in the actual step (Figure 7.5).

In the following sections the utilization of the panels will be described in detail together with the available options.



Figure 7.5: Navigation buttons

7.1.1 A workload characterization study

In this section the main operations to be performed in a workload characterization study are described. The definition of the *workload basic components*, that is, the identification of the smallest level of detail that is to be considered and of the set of variables to be used for their description, is among the first operations to be performed. Depending on the intended use of the study, as workload basic components one may select interactive commands, database queries and updates, web site requests, web downloads, programs, scripts, servlets, applets, traffic intensity, source level instructions, etc. In the following we will refer to a workload basic component as an *observation* (i.e., an item in the input file) and to the parameters that describe each component as *variables*.

1. The first step consists of the selection of the file containing the data that are to be considered as input, of the variables to load and of their format (see section 7.1.2). Let us remark that each observation may be described in the input file by n variables but that the number of variables that are considered in the actual statistical analysis may be less than n. It is also possible to decrease the number of observations of the original input file (decrease its size) that will be loaded (considered) in the characterization study. Several filtering criteria for the construction of a sample of data are available: random selection of a given number of observations, selection of the observations that belong to a given interval of their sequence number. At the end of the data loading a panel describing the errors found in the data is shown. A user may decide to stop the analysis and verify the input data or may continue with the following steps.
2. Computation and visualization for each variable of the descriptive statistics univariate and bivariate, of the frequency graphs, of the scatter plots, and of the QQ-plot compared to a normal or exponential distribution (section 7.1.4).
   It is important to observe that usually the variables are expressed in non-homogeneous units. To make them comparable a *scale change* must be performed. The following statistical analysis of the original data, especially of the histograms of their parameters, allows us to determine when it would be useful to apply to them a logarithmic or another transformation. Very often the density functions of some variables are highly positively skewed and have high values of kurtosis too. Applying a logarithmic transformation of highly skewed variable densities it is possible to avoid or to reduce correlations among means and variances of the clusters that will be identified. Several *transformations* may be applied to the numeric variables (attention, *ONLY* to the numeric variables) .
   A *sample* may be extracted from input file by using different criteria. The distributions of two variables may be compared through QQ-plots and scatter plots selected in the corresponding matrix.
3. Selection of the clustering algorithm to apply and of the variables to be considered in the cluster analysis (see section 7.1.5). The basic idea is to partition the given set of observations into classes of components having homogeneous characteristics. Among the various clustering algorithms the most commonly applied to workload characterization problems belong to the non-hierarchical k-means family. The algorithm produces a number of classes ranging from 1 to k, where k is a parameter that must be specified by the user. The observations are assigned to a class so that the sum of the distances between all the elements and the centroid of the class is minimized. A set of representative components will then be selected from the classes according to some extraction criteria (usually the centroids of the clusters) and used in the construction of the workload model.
4. A panel is available for the visualization of the results of the clustering analysis. Several graphics and tables are reported in order to describe the clusters composition and the goodness of a partition. Scatter plots are available for all the variables considered (see section 7.1.6).

7.1.2 Input definition

The panel for the description of the input data (Figure 7.2) is a very important one since it allows the selection of the input file that contains the data to be analyzed and the description of their structure. In such a way we may take into consideration the log files generated by different systems and tools. The format definition structure of the input file is described in detail in Section subsection 7.1.3.

Input file selection

To select the input file use the panel shown in Figure 7.6. The *Browse...* option for the selection of the input file is available (Figure 7.6). Once a file has been selected its name is added to the list of the files available for following analyses. A panel showing the number of observations loaded and the file name will automatically

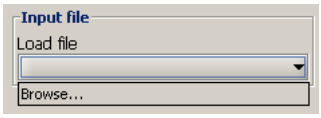


Figure 7.6: Panel for the selection of the input file

appears (Figure 7.7).



Figure 7.7: Panel with information about the selected file

Format definition

The next operation consists of the definition of the data format. The format of the input data is required by the application:

- to recognize the number of variables that belongs to each observation
- to recognize the type (string, numeric or data) of each variable
- to the right interpretation of the meaning of each data element that belong to a single observation. For the definition of the elements regular expressions Perl 5 (subsection 7.1.3) are used. The defined formats are stored in a library and can be browsed and reused.
- to select a subset of the variables of the observations that are in the original input file.

We will describe a very simple example of the format definition of an *observation*, i.e., a single element in the input log file. The considered log file contains several data, e.g., IP client address that submits the request for a resource to a web server, request timestamp, etc...) that are called *variables*.

After the selection of the input file, the format definition table is empty, see Figure 7.2, since no information regarding current format has already been defined.

Two types of format definition are available:

- Manual, that consists of the definition of the format of each single variable through relative table and relative commands
- Automatic, that load one of the predefined standard formats previously defined and loaded in files *'.jwat-format'*.

Manual definition

At the beginning, the correct number of variables of each observation in the log file is to be set (use the spinner *'Number'*). By using the '*New var*' button a new variable is added to the format table. The same result is obtained using the spinner *'Number'* at the right top of the table (see Figure 7.8).

To remove one or more variables by a format table we can utilize:

- the button  at the end of each row of the table. The result is the deletion of the concerning variable
- turned arrow towards the bottom of the spinner. The result is the deletion of the last variable of the table
- the right button of the mouse on a single selected variable, or on a set, to delete them from the format table (Figure 7.9).

Once the correct number of variables in a observation is defined the next step is their description. The columns of the format table have the following meaning:

- *Name:* as default, to each new added variable it is assigned the standard name *Variable X*. In order to facilitate the analysis of the results it is recommended to change these names with new ones related with the meaning of the variables.

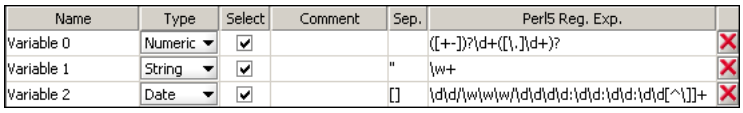


Figure 7.8: Format table

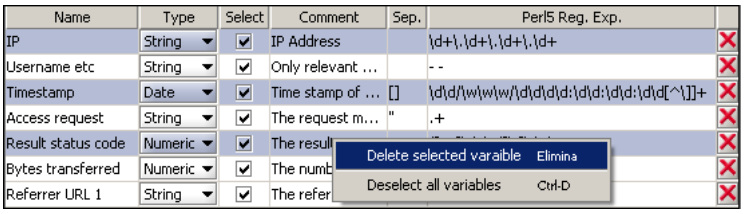


Figure 7.9: Deletion of selected variables

- *Type:* it identifies the type of the variable. Possible choices are *number, string* and *data*. The type definition is fundamental for the right interpretation of the results.
- *Selection:* it allows to specify which of the variables of the observations registered in the input file will be effectively loaded. In any case, it is necessary to define all the variables of each observation, also if some of them will not be selected for the statistical analysis.
- *Comment:* it is allowed to add a string of comment to each variable.
- *Delimiter [Sep]:* it identifies the character used (if it exists) in the input file to delimit the field identified by the current variable. For example in the Apache server log file the variable "timestamp" is contained within square brackets.
- *Regular Expression :* definition of regular expression (Perl 5) that defines the variable. For example in case of IP variables a possible regular expression is: '$\backslash d+\backslash.\backslash d+\backslash.\backslash d+\backslash.\backslash d+$'.
- *Delete:* the corresponding variable will be deleted.

In order to help the users for each of the types the tool sets automatically some of the fields (delimiter of field and regular expression) supplying standard settings as shown in Figure 7.8 where for for the type *string* it is proposed " as delimiter and \w+ as regular expression.

A format definition may be saved by using the '*Save format*' button so that it may reused in the future.

Format loading

If a standard log file (Apache, IIS) is used it is possible to load the corresponding format by using the combobox in the panel of Figure 7.10.

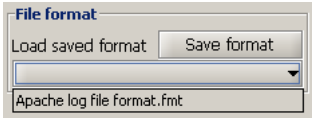


Figure 7.10: Panel for the loading of standard and previously saved format

For example, by selecting the option "Apache log file format", the format definition table of Figure 7.11 is automatically loaded. At this point, the user can select through the format table selection field the subset of variables to load.

Methods for extracting a sample

In order to process the data from real log files, that usually are constituted by a very large number of observations (e.g., several Gigabytes), by a cluster algorithm with a reasonable amount of processing time and storage dimensions, the size of the original data set has to be reduced. Thus, only a sample drawn from the original set of data is often submitted as input to the clustering algorithm.

After the definition of the data format it is possible through the panel *Workload sampling method* of Figure 7.12 to select one of the following extraction criteria:

- *Complete file:* all the observations of the input file are considered.
- *Random:* the number of observations to be loaded should be specified in the panel of Fig.Figure 7.13.
- *Interval:* the panel of Figure 7.14 that requires to specify the interval of observations (according to their id numbers) to load will be shown.

Name	Type	Select	Comment	Sep.	Perl5 Reg. Exp.	
IP	String	☑	IP Address		\d+\.\d+\.\d+\.\d+	✕
Username etc	String	☑	Only relevant ...		- -	✕
Timestamp	Date	☑	Time stamp of ...	[]	\d\d/\w\w\w/\d\d\d\d:\d\d:\d\d:\d\d[^\]]+	✕
Access request	String	☑	The request m...	"	.+	✕
Result status code	Numeric	☑	The resulting s...		([+-])?\d+([\.]\d+)?	✕
Bytes transferred	Numeric	☑	The number of ...		([+-])?\d+([\.]\d+)?	✕
Referrer URL 1	String	☑	The referring p...		[^\s]+	✕
Referrer URL 2	String	☑	The referring p...	"	.+	✕
User Agent	String	☑	The user agent...	"	.+	✕
Referrer URL 3	String	☑	The referring p...	"	.+	✕

Click or drag to select classes; to edit data single-click and start typing. Right-click for a list of available operations

Figure 7.11: Format definition table

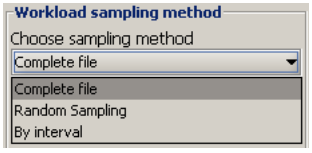


Figure 7.12: Extraction criteria for the sample construction

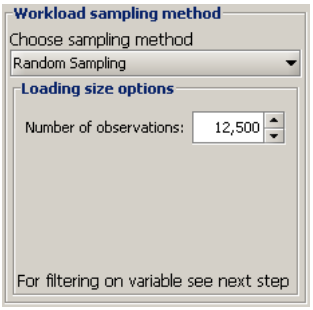


Figure 7.13: Random method for the construction of a sample

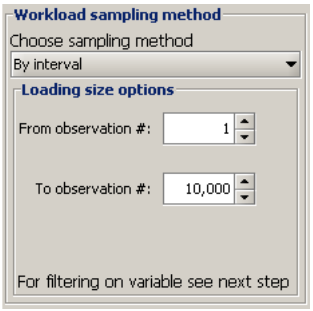


Figure 7.14: Interval method options

Loading of log file

After the definition of the input file the loading of the selected observations will be started. After pressing the '*LOAD*' button (Figure 7.15) the loading window of Figure 7.16 will be opened. The percentage of loading completion, the number of processed observations and the number of observations discarded because not consistent with the specified format are shown. At the end of the load a window summarizing the total number



Figure 7.15: Button for the loading of the input data.

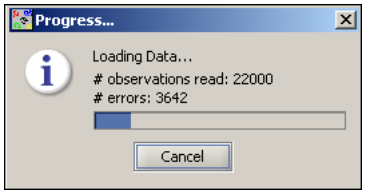


Figure 7.16: Window showing the loading progress

of observations loaded and the total number of observations loaded correctly (Figure 7.17) is shown. It is

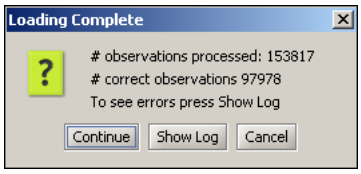


Figure 7.17: Loading phase results

possible to visualize the log file that contains the description of the errors found during the loading operation (Figure 7.18). The error messages that are contained in the log file are:

- *Error in row xx : Element y is wrong*: (if you defined separators) the element corresponding to the variable y, for the row xx, does not correspond to the defined regular expression.
- *Error in row xx : Line does not match format (element y not found)*: The element corresponding to the variable y is not present in the row (observation) xx of the input file.
- *Error in row xx : Too many fields* : the row xx contains elements in excess compared to the defined format. This may indicate with high probability that the format definition described by the user is not correct for the considered observation.

7.1.3 Format definition

In this section we will describe the format definition of the file in input to the application. We will introduce the basic concepts on which the format specific is based and then we will describe some examples with different files.

The application allows, by using regular expression, to utilize as input either log files of standard formats, as Apache and IIS, or log files modified by the user. The only important requirement on the input file structure is that every observation (i.e., the set of the variables that refer to one element/item considered) must end with '\n' character (new line).

To allow the application to be able to process correctly a log file, the user must describe:

- the observation structure, that is the number of variables that compose it.
- and for each variable the name, the type and the 'structure'.

These information are specified by the user through the input panel described in section 7.1.2.

Regarding the number of variables it is enough to insert in the format table a number of rows equal to the number of elements that compose the observation (see section 7.1.2). In Figure 7.19 the observation is composed of 6 variables.

The definition of the single variable requires, in addition to the name, a comment, its selection in order to be considered in the analysis or not, its type, the description of the possible delimiter (Separator) and of the regular expression that characterizes it.

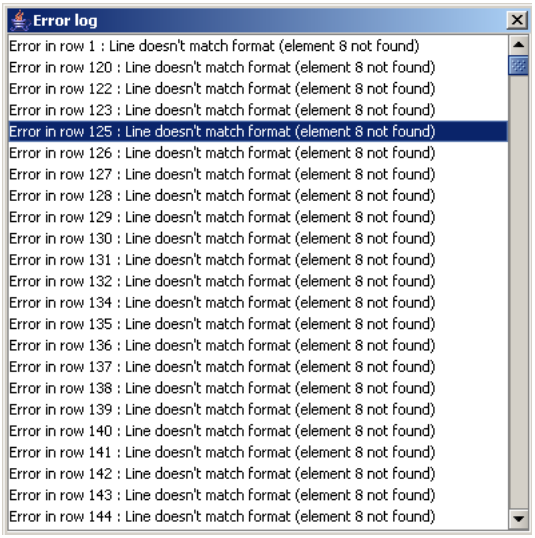


Figure 7.18: Example of error log in the loading phase

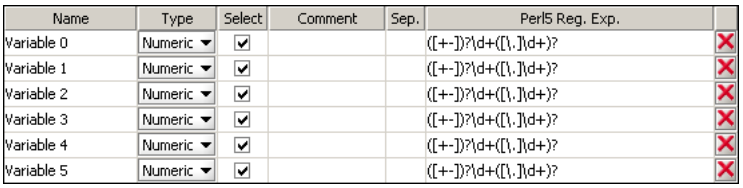


Figure 7.19: Format table after the insertion of the number of variables

The type has to be chosen among the *numeric, string o data*. The delimiter defines (if it exists) the character or the characters that delimit the variable's value. The regular expression (Perl 5) defines the format with which the variable value is recorded in the log file.

A concise description of the format used for the regular expressions (Perl 5) follows. The purpose of the definition of the regular expression is to specify the pattern that the variable values must have and therefore (together with selected type) to allow the application to process correctly its value. For a complete description of the regular expressions see a Mastering regular expressions book. In this section we will describe some options useful to define in a simple way the pattern of the variables values. A list of operators follows:

+ : it indicates that there are one or more instances of the preceding character (a+ ⇒ a,aa,aaa,...)

[ ]: it defines a pattern that allows to utilize as match element one of the values within square brackets (a[bcd]e ⇒ abe, ace, ade)

***** : it indicates that there are zero or more instances of the preceding character (ba* ⇒ b,ba,baa,baaa,...)

? : it indicates that there are zero or one instance of the preceding character (ba? ⇒ b,ba)

^ : it has two different meanings depending on the use. The former permits to require that a pattern starts with a particular character (^abc ⇒ abcdef,abc fg,...). If it is utilized within square brackets as first character, it allows to match any character that doesn't belong to the defined group ([^a] ⇒ bcd,wedfgr,...)

() : it defines a pattern that permits to use as match element the string that is contained between brackets ((ab)+ ⇒ ab,abab,ababab,...).

A list of *Escape Sequences* used to identify intervals of characters follows:

\d : it identifies whichever number [0-9].

\D : it identifies whichever thing except a number [^0-9].

\w : it identifies whichever character or number [0-9a-zA-Z].

\W : it identifies whichever thing except a character or a number [^0-9a-zA-Z].

\s : it identifies a *white space* character [\t\n\r\f].

\S : it identifies whichever thing except a *white space* character [^\t\n\r\f].

. : it identifies whichever character except the *new line* character '\n'.

Combining in an opportune way the characters, the options and the *Escape Sequences* it is possible to define correctly the regular expression for each variable.

In the specific of regular expressions in Perl 5 there are some characters that are considered *metacharacters*. Some of them are the same operators defined before. In this case, to use such characters as elements in the expression is enough to precede them with the \ character.

The delimiter identifies one or more characters that delimit the value of a variable. In the log file of the Apache format the page URL requested by the user contains characters and spaces and is delimited by “”. A simple definition of the $\backslash w+$ type regular expression could produce a wrong result because the value recovered by application would finish at the first met space and not at the closure ”. To avoid this problem it is possible to utilize the delimiters so that by using the expression $.+$ and specifying as delimiter ” the result will be correct.

Concerning the Apache log file, and more precisely the *timestamp* field, it is possible to show another use of the delimiter. The value is contained between characters ‘[’ e ‘]’ (e.g, *[12/Dec/2001:00:00:03 +0100]*) and by using the field delimiter it is possible to specify as regular expression the following:

$$\backslash d\backslash d/\backslash w\backslash w\backslash w/\ \backslash d\backslash d\backslash d\backslash d:\backslash d\backslash d:\backslash d\backslash d:\backslash d\backslash d$$

This expression doesn't specify the last part of the field (*+0100*) but it is correct in any case because specifying the delimiters it is possible to define the regular expression in order to match only a part of the variable value.

Example of a format definition

Let's consider a file to be analyzed with the following structure of the observations:

String without spaces 1000 0.0876 -12.663 “String with spaces” [12/Dec/2001:00:00:03 +0100]

The variables are 6 and their definition (Figure 7.20) is the following:

String without spaces : we can define the regular expression in different ways, as for example *[a-zA-Z]+* if the string doesn't contain numbers or $\backslash w+$ to indicate whichever string of numbers and characters.

Integer number : we define the regular expression as a sequence of one or more numbers and therefore as $\backslash d+$.

Positive decimal : in this case there is the possibility to have decimal digits so that we define the regular expression as a sequence of one or more digits ($\backslash d+$) that may have the decimal separator and a further succession of one or more digits. The result is therefore $\backslash d+(\lceil\backslash .\rfloor\backslash d+)?$.

Decimal with sign : in this case in comparison to the previous case there is the possibility that at the beginning of the number there is a sign +, - or no sign. Therefore it is enough to add at the beginning of the regular expression *([+-])?*.

String with spaces : if we utilize the regular expression previously defined $\backslash w+$ we would have a problem because the value recovered from the log file would be ”String that is not correct. The best solution is to utilize the delimiter, in this case ”, and to utilize therefore the definition $.+$.

Date : even in this case by using the delimiter (*[]*) we can define the following regular expression \d\d/\w\w\w/ \d\d\d\d:\d \d:\d\d:\d\d that may be used to recover the only information “12/Dec/2001:00:00:03” leaving out the remaining part.

Name	Type	Select	Comment	Sep.	Perl5 Reg. Exp.	
Stringa senza spazi	String	☑			\w+	✗
Numero intero	Numeric	☑			\d+	✗
Numero Reale	Numeric	☑			\d+([\.]\d+)?	✗
Reale con segno	Numeric	☑			([+-])?\d+([\.]\d+)?	✗
Stringa con spazi	String	☑			(\w+[\s]?)+	✗
Data	Date	☑		[]	\d\d/\w\w\w/\d\d\d\d:\d\d:\d\d:\d\d[^\]]+	✗

Figure 7.20: Example of the definition of a file format

Example of the definition of the Apache log

Now we will describe the format of the Apache log. Each observation has the following structure:

151.29.12.105 - - [12/Dec/2001:00:00:10 +0100] "GET /mappe HTTP/1.1" 200 313 www.polimi.it "http://www.polimi.it/" "Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)" "-"

A brief explanation of the regular expressions used (Figure 7.21) follows:

1. for the Ip address of the request we define a sequence of one or more digits separated by a point and therefore the regular expression is *\d+\.\d+\.\d+\. \d+*.
2. in most cases no value is set, and therefore it is possible to define exactly the string that we want to recover from the file through - - (it is advisable not to select this variable at the moment of loading).
3. for the date we use the same definition described in the previous example or more simply by selecting the date type in the concerning box, the application supplies a default definition that is correct too.
4. it is a string, delimited by ", that can contains within some spaces characters and numbers. The best definition uses the delimiter and the regular expression *.+*.
5. it is an integer number and as we have seen before we can utilize a more simple regular expression *\d+* o *([+-])?\d+([\.] \d+)?*.
6. even in this case it is an integer number and as we have seen before we can utilize the regular expression *\d+* o *([+-])?\d+([\.] \d+)?*.
7. in this case the string is not delimited by any particular character and it doesn't contain any spaces so that we can utilize a simple regular expression *\w+* or ones little bit different as *[^\s]+*.
8. these last three strings are all delimited by the character " and they can contain some spaces, characters or numbers and therefore we must utilize the regular expression *.+*.

Name	Type	Select	Comment	Sep.	Perl5 Reg. Exp.	
IP	String	☑	IP Address		\d+\.\d+\.\d+\.\d+	✕
Username etc	String	☑	Only relevant ...		- -	✕
Timestamp	Date	☑	Time stamp of ...	[]	\d\d/\w\w\w/\d\d\d\d:\d\d:\d\d:\d\d[^\]]+	✕
Access request	String	☑	The request m...	"	.+	✕
Result status code	Numeric	☑	The resulting s...		([+-])?\d+([\.]\d+)?	✕
Bytes transferred	Numeric	☑	The number of ...		([+-])?\d+([\.]\d+)?	✕
Referrer URL 1	String	☑	The referring p...		[^\s]+	✕
Referrer URL 2	String	☑	The referring p...	"	.+	✕
User Agent	String	☑	The user agent...	"	.+	✕
Referrer URL 3	String	☑	The referring p...	"	.+	✕

Figure 7.21: Apache log file format definition

7.1.4 Data processing

A tabs substructure (Figure 7.22) shows the groups of the various statistics available:

Univariate: descriptive statistics, graphs and transformations.

Sample construction: methods of sample construction and data filtering.

Bivariate: bivariate statistics.

QQ-plot matrix: QQ-plots of the variables for comparison with normal and exponential distributions

Scatter plot matrix: Scatter plots of all the variables.



Figure 7.22: Tabs structure for the processing of the data

Variable	Mean	Variance	Std. Dev.	Coeff. of var.	Minimum	Maximum	Range	Median	Kurtosis	Skewness	Num. Obs.
IP	330.431E0	687.347E2	262.173E0	793.426E-3	000.000E0	908.000E0	908.000E0	164.000E0	-880.929E-3	713.959E-3	358.380E2
Timestamp	100.839E10	225.805E12	150.268E5	149.018E-7	100.837E10	100.841E10	417.500E5	100.839E10	-166.636E-2	-180.260E-4	358.380E2
Access request	333.612E1	185.306E5	430.471E1	129.034E-2	000.000E0	142.380E2	142.380E2	735.000E0	-441.802E-4	116.237E-2	358.380E2
Result status...	202.568E0	390.938E0	197.722E-1	976.073E-4	100.000E-2	404.000E0	403.000E0	200.000E0	748.764E-1	820.835E-2	358.380E2
Bytes transfe...	492.681E1	381.579E6	195.340E2	396.485E-2	000.000E0	103.972E4	103.972E4	956.000E0	646.421E0	200.840E-1	358.380E2
Referrer URL 1	195.323E-6	418.524E-6	204.579E-4	104.738E0	000.000E0	300.000E-2	300.000E-2	000.000E0	157.631E2	120.562E0	358.380E2

Figure 7.23: Descriptive statistics of the variables

	IP	Timestamp	Access request	Result status code	Bytes transferred
IP		402.181E-13	193.853E-9	124.385E-7	191.049E-10
Timestamp	402.181E-13		717.968E-14	487.647E-12	723.455E-15
Access request	193.853E-9	717.968E-14		584.537E-8	718.559E-11
Result status code	124.385E-7	487.647E-12	584.537E-8		136.254E-8
Bytes transferred	191.049E-10	723.455E-15	718.559E-11	136.254E-8	

Figure 7.24: Correlation coefficients of the variables

Statistics

The univariate and bivariate panels show respectively several statistical indexes concerning each variable of the observations (Figure 7.23) and the correlation coefficients of the variables (Figure 7.24).
The univariate panel shows also the following two types of graphs:

- a histogram, or frequency graph, preview for each variable, that can be magnified with a double click on the preview (Figure 7.25).

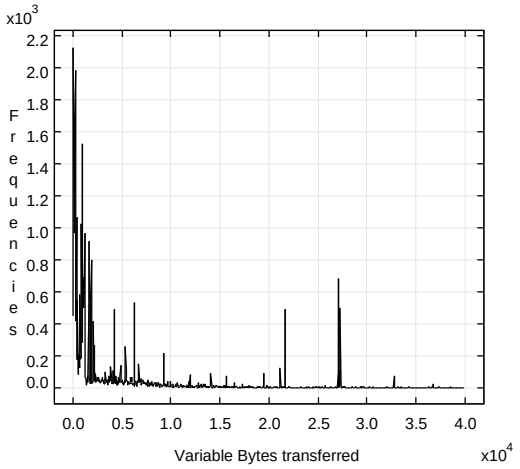


Figure 7.25: Histogram, or frequency graph, of the variable "Bytes transferred"

- a preview of the QQ-plot graph of the variable quantiles compared with the normal distribution quantiles with the same average and variance or the exponential distribution with the same average. Also this preview can be magnified with a double click (Figure 7.26).

Both magnified graphs can be saved in *eps* (Encapsulated PostScript) and *png* (Portable Network Graphics) formats.

Transformations and sample extraction

The panel of univariate statistics allows also to perform some variables transformations. To apply a transformation select the variable in the table of Figure 7.23; if it is of a numeric type the panel of Figure 7.27 that allows the selection of the type of transformation (logarithmic, min-max and standard deviation) will be shown and press the '*Apply transformation*' button. It is possible to undo the last transformation by pressing the '*Undo transformation*' button. The list of the variables with the transformations applied is reported in the table. If the selected variable is not numeric, the following message is shown: *"This variable could not be transformed since it is not numeric"*.

To extract a sample from the original input file press on tab *'Sample construction'*. The panel of Figure 7.28 shows the possible criteria that can be applied to the loaded variables that depends on the selected method and on the type of the variables.

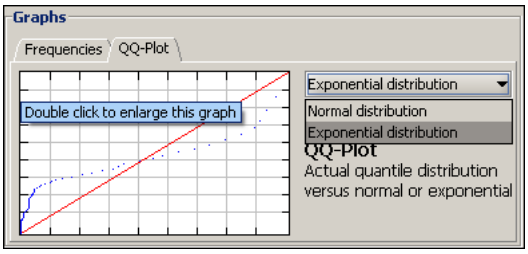


Figure 7.26: QQ-plot preview for the comparison of a given distribution with an exponential or a normal ones.

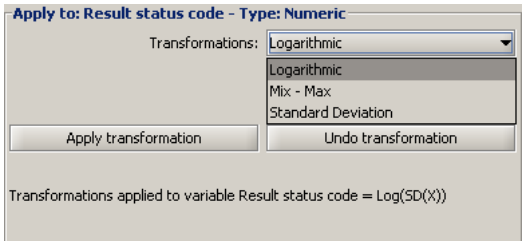


Figure 7.27: Selection of the transformation to apply to the original data

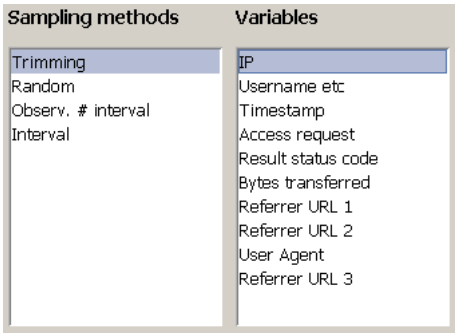


Figure 7.28: Sampling extraction criteria that can be applied on the selected variables

- *Random* and *Obsev. # interval* methods are similar to those available in input window and allow to select randomly n observations from the input file or to extract the observations whose numerical identifiers are included into the interval defined by the user.
- *Trimming :* the outliers, i.e., those values of a variable that are too distant from the other values of the same variable, may distort the transformation and the statistical analysis by causing the other observations to be assigned too much or too little weight should be removed. To filter a variable distribution (trimming operation) the percentiles that should be removed can be specified (Figure 7.29).

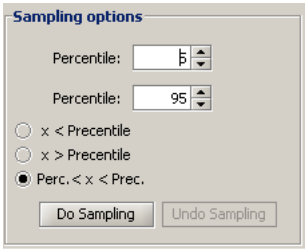


Figure 7.29: Filter of the percentiles of a variable (trimming of the distribution)

- *Interval* operation applies a filter on a variable's values. Depending on the type of the variable different options panels will be shown. If the selected variable is *numeric* the panel is similar to that of Figure 7.30: the minimum and maximum values of the variable should be specified. If the selected variable is of *data* type the panel is similar to that of Figure 7.31: the dates of start and end of observations should be specified. If the variable is of *string* type, it is possible to specify a substring that has to be contained in the selected observations.

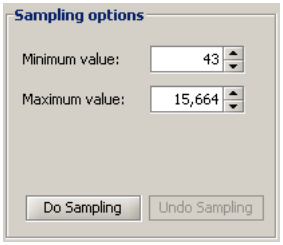


Figure 7.30: Filter on the values of a numeric variable

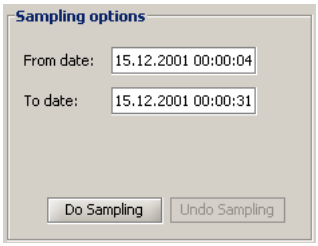


Figure 7.31: Filter on the values of a data variable

The operation of *'Undo sampling'* is available.

Graphs

The last two panels of the statistics tab concern a preview of QQ-plot graphs and of scatter plots for all the variables, respectively. Both allow to save graphs in the most common formats(,). Previews can be magnified with a left double click of the mouse. Several functions are available for the graphs: portions zoom, points dimension, by clicking the right button of the mouse on a graph various formats are available for export the graph as image.

7.1.5 Clustering algorithms

The next step of the characterization process is the choice of the clustering algorithm to be used ad of the options available. Two clustering algorithms are actually implemented in the tool (k-Means and Fuzzy k-Means), see

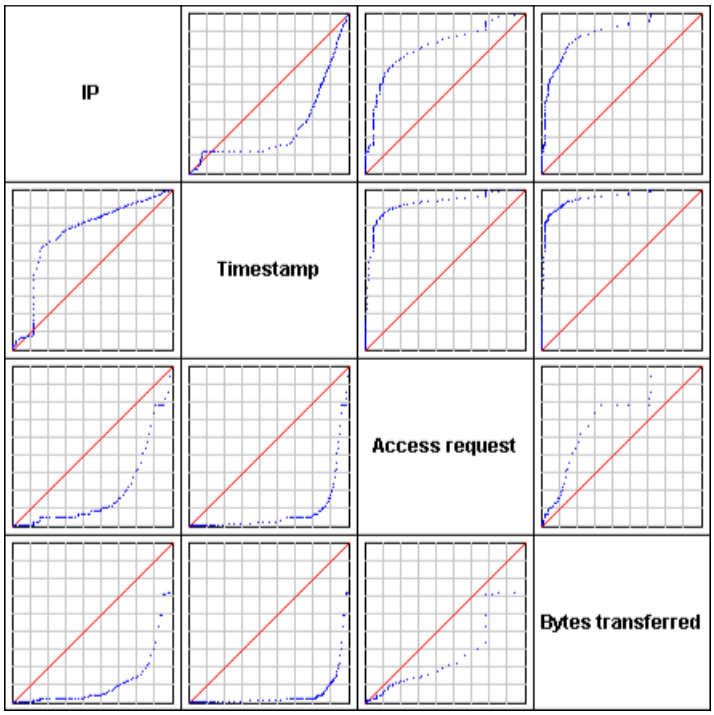


Figure 7.32: QQ-plot matrix for the comparison of the distributions of two variables

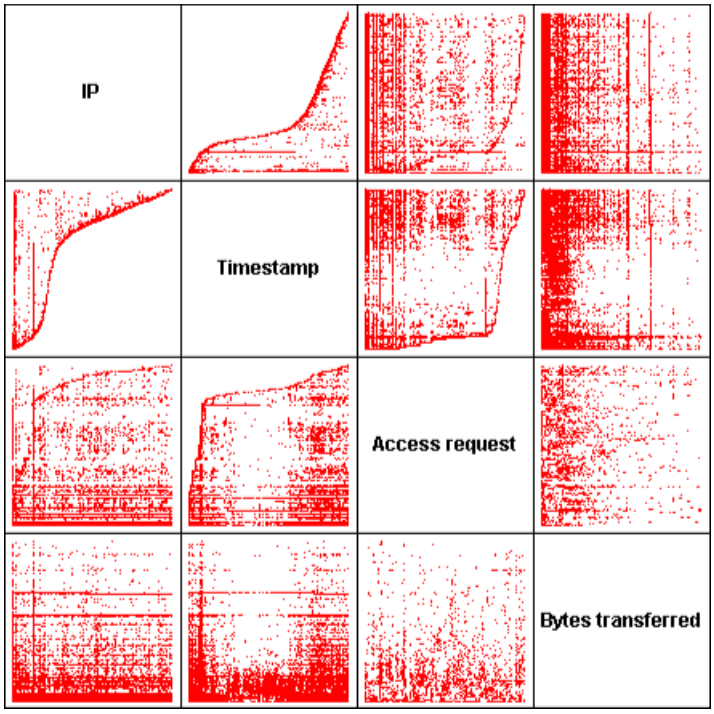


Figure 7.33: Scatter matrix

Figure 7.34. Depending on the selected algorithm, in the right panel of tab the available options are listed.

The execution of a clustering algorithm will start when the buttons '*Solve*' or  are pressed. During the execution, the window of Figure 7.35 that report the progress state and allows to interrupt the execution is shown.

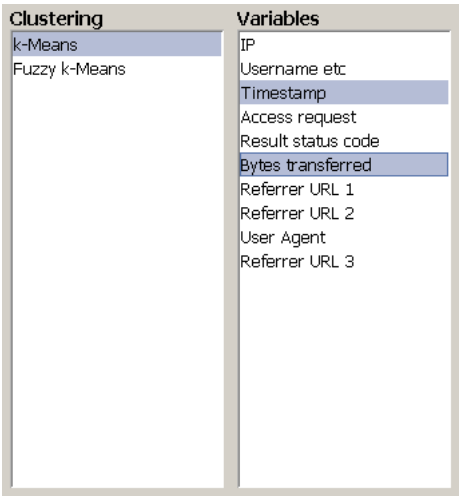


Figure 7.34: Clustering algorithm used and variables selected that will be considered in the analysis.

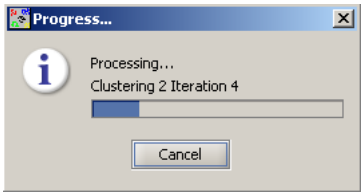


Figure 7.35: Clustering progress panel

K-Means algorithm

The non-hierarchical k-Means algorithm implemented in the tool requires the specification of the following parameters, Figure 7.36:

- *the maximum number k of clusters* that the algorithm has to produce. Note that the algorithm produces the results for all partitions from 1 to k clusters
- *the maximum number of iterations* to perform in order to find the optimal partition. The initial subdivision into k clusters is iteratively improved by shifting, based on the selected criterion (in this case the Euclidean distance), the elements of a cluster to another and computing after each assignment the new center of mass of the clusters. The optimum configuration is obtained when points can no longer be reassigned. The selected value is an upper limit of the number of interactions for each partition. Experiences suggest the value of 4 interactions as a reasonable choice: the results are obtained in a short time and are enough accurate. To obtain more accurate results higher values should be used, this involves a higher computation time
- *the transformation type* to apply to selected variables (if needed). Transformations of the values are applied before the execution of the algorithm and at the end of the execution the results can be transformed back to their original values. The transformation of the values is often required since the variables are usually expressed in different units and their ranges are very different. Since the algorithm uses the Euclidean distance function as comparison metric to determine if an observation belongs to a cluster, the results could be not reliable if the values differ of one or more order of magnitude.

Fuzzy k-Means algorithm

The fuzzy k-Means algorithm implemented in the tool requires the following parameters, Figure 7.37:

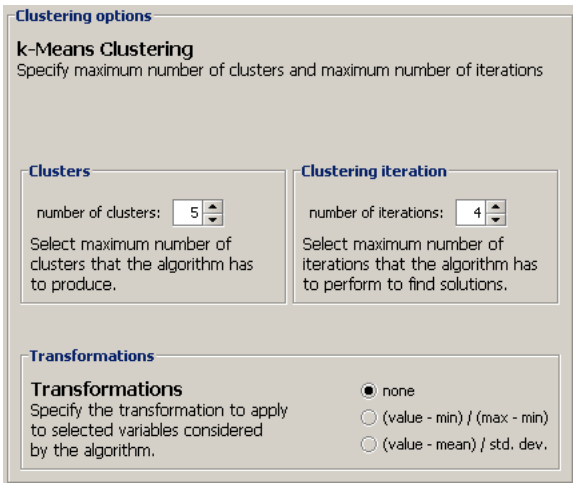


Figure 7.36: Parameters of the clustering algorithm k-Means

- *the maximum number k of clusters* that the algorithm has to produce. The results for all the partitions from two to the selected maximum value will be provided. The higher is this value the higher is the execution time of the algorithm
- *the maximum number of iterations* to perform in order to find the optimal partition. Four is a suggested value. See the comments reported above for the K-means algorithm.
- *the fuzziness level* to apply during the algorithm execution. It is the value used by algorithm to determine the fuzziness degree of final solution. It ranges from 2 to 100
- *the transformation type* to apply to selected variables (if needed). See the comments reported above for the K-means algorithm.

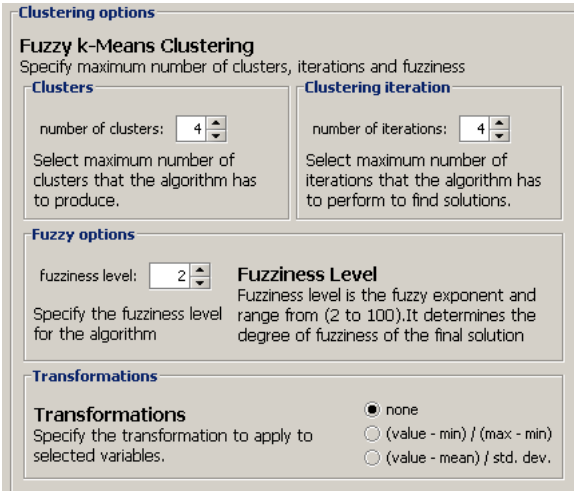


Figure 7.37: Parameters of the clustering algorithm Fuzzy k-Means

7.1.6 The results of a clustering execution

At the end of the execution of a clustering algorithm the tab that allows the analysis of the results is shown. The main panel shows the table in Figure 7.38 that contains the algorithm(s) executed and the maximum number of clusters identified. It is possible to delete the results of an execution by clicking the button  in the last column.

Depending on the selected clustering algorithm different results panels are shown in the results table.

K-Means algorithm results

When the $k - Means$ algorithm results are selected, a table reporting the indices concerning the goodness of the partitions is shown, Figure 7.39.

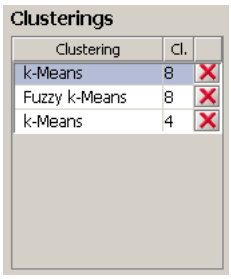


Figure 7.38: List of executed clustering algorithms

To estimate the optimal number k of clusters in which the given set of observations will be subdivided two indicators of the goodness of a partition are used. For each variable the *Overall Mean Square Ratio, OMSR*, i.e., a measure of the reduction of within-cluster variance between partitions in k and in $k+1$ clusters, and the *ratio* of the variance among the clusters and the within-cluster variance have been used. Large values of the $OMSR$ justify increasing the number of clusters from k to $k+1$. An optimal partition should also be characterized by values greater than 1 of the *ratio*s of the variances among the clusters and within the clusters for each variable.

A visual indication of the goodness of a partition is given with the icons and representing good and bad results respectively. The values of OMSR and of the *ratio* are also reported in the table.

By selecting one of the rows of the table in Figure 7.39 the panel is updated and in the right side a tabs structure concerning the visualization of the results is shown. Three main panels are available (Figure 7.40):

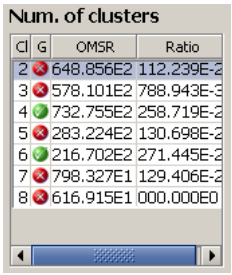


Figure 7.39: Indicators of the goodness of a partition (*Overall Mean Square Ratio* and *ratio* indices) for the K-Means algorithm

- *Clustering Info:* It shows some statistical information concerning the clusters. For each cluster the number of its components and its weight with respect to the total population are reported. Pie-charts are also used to visualize these data. The panel at the bottom shows for each variable the distribution (in %) of its values among the clusters.

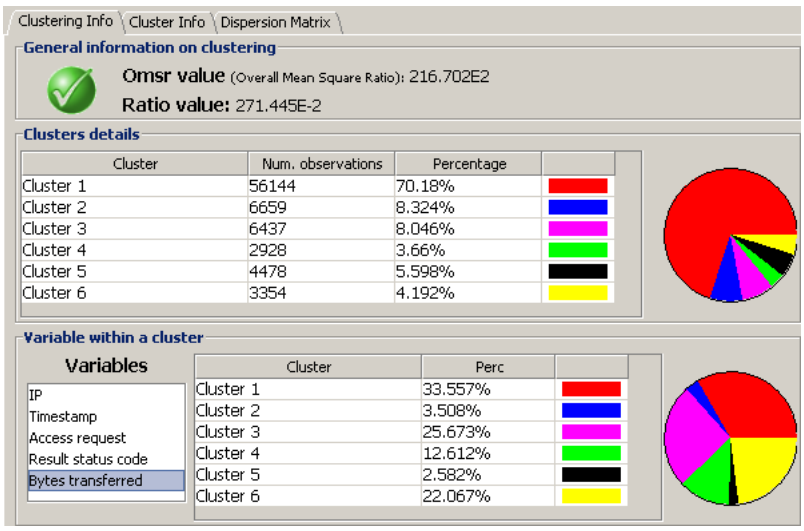


Figure 7.40: Characterization of the clusters

- *Clusters Info:* It shows statistics concerning the clusters (Figure 7.41). Besides the classic descriptive univariate statistics, the *ISC*, that indicates which variables have been used for the clustering, and the *center* values, that refer to the centroid coordinates of each cluster, are reported. The panel at the bottom shows a preview (it can be magnified by a left double click of the mouse) of the graph concerning the comparisons of the distributions of the variables used in each cluster. It is also possible to visualize the list of observations that belong to a cluster through the *'Show Observations'* button (attention! this operation may requires several minutes).

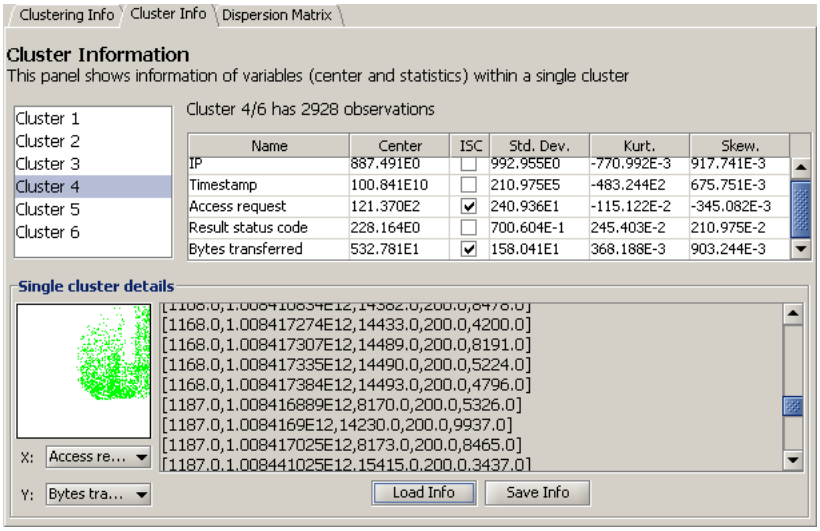


Figure 7.41: Descriptive statistics of the clusters obtained with the k-means algorithm and their visual characterization through the dispersion matrix

- *Dispersion Matrix:* this panel shows the matrix of all the possible variable *vs* variable scatter plots (Figure 7.42). These graphs allows a visual interpretation of the compositions of the clusters since the observations are represented with the color corresponding to the cluster it is assigned. With a left double click of the mouse on a preview, the graph is magnified and several options can be selected with a right click on the graph.

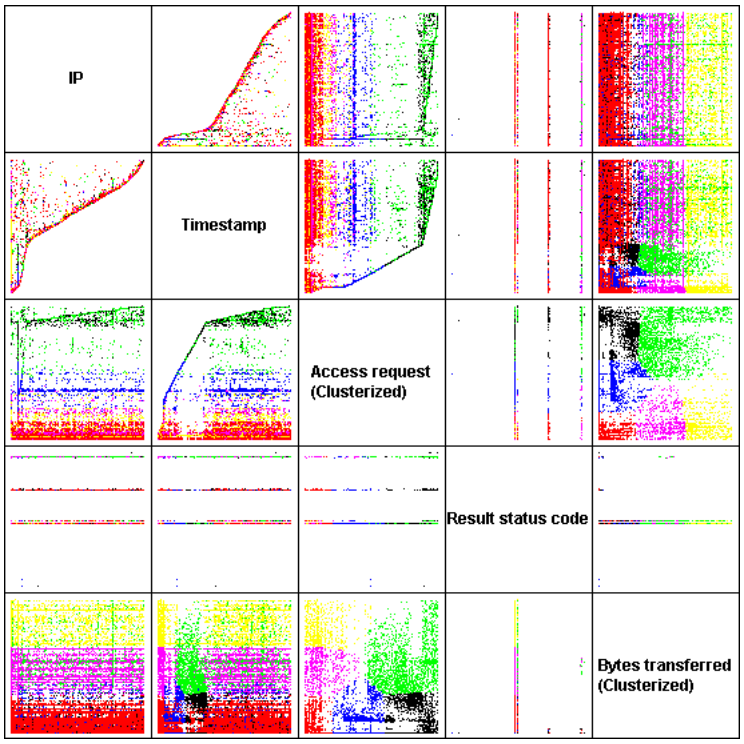
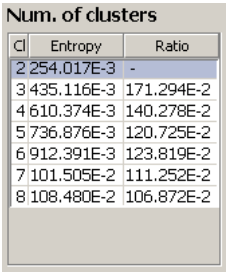


Figure 7.42: K-Means scatter plots matrix. Each observation is represented with the color corresponding to the cluster it is assigned.

Fuzzy K-Means algorithm results

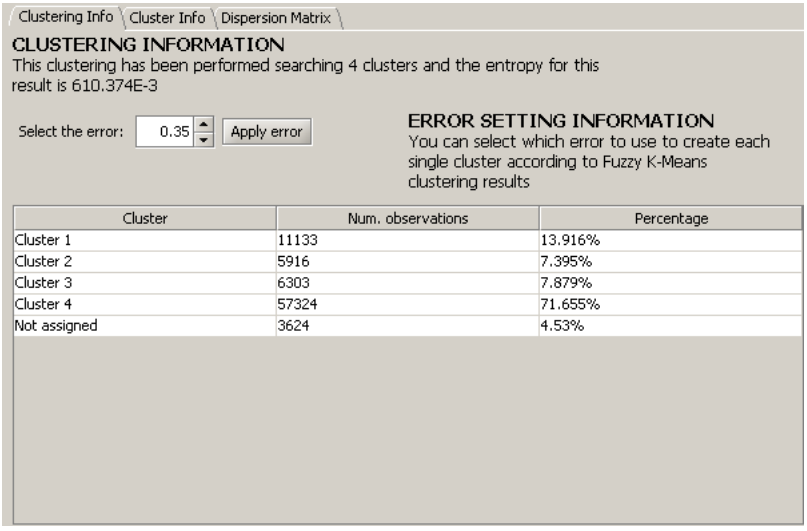
When the fuzzy k-Means results are selected, the panel is uploaded and in the table of Figure 7.38 a new table containing a row for each partition identified is shown (Figure 7.39). The number of clusters, the *Entropy* index and the *ratio* are shown. By selecting one of the rows of table in Figure 7.43 the panel is updated and appears in the right side a tabs structure that permits to visualize information concerning results, by grouping them in three main panels:


Num. of clusters

Cl	Entropy	Ratio
2	254.017E-3	-
3	435.116E-3	171.294E-2
4	610.374E-3	140.278E-2
5	736.876E-3	120.725E-2
6	912.391E-3	123.819E-2
7	101.505E-2	111.252E-2
8	108.480E-2	106.872E-2

Figure 7.43: Results of a fuzzy k-Means algorithm execution

- *Clustering Info:* It shows the clusters that have been identified and the entropy values (Figure 7.44). The error value used to identify a partition should be set by the user and can be easily changed (error setting information are reported on the right). Pressing the '*Apply error*' button the assignment of the observations in the current partitions will be changed according to the new error value and the table at the bottom showing the composition of each cluster (and the statistical and graphic information) of the following panels will be updated. Note that the sum of the percentages could be *greater* than 100% because of the multiple assignments of the fuzzy algorithm.


Clustering Info | Cluster Info | Dispersion Matrix

CLUSTERING INFORMATION

This clustering has been performed searching 4 clusters and the entropy for this result is 610.374E-3

Select the error: 0.35 Apply error

ERROR SETTING INFORMATION

You can select which error to use to create each single cluster according to Fuzzy K-Means clustering results

Cluster	Num. observations	Percentage
Cluster 1	11133	13.916%
Cluster 2	5916	7.395%
Cluster 3	6303	7.879%
Cluster 4	57324	71.655%
Not assigned	3624	4.53%

Figure 7.44: Results of a fuzzy k-means algorithm execution

- *Clusters Info:* It shows statistical information concerning the selected cluster (Figure 7.45). See the comments reported above for the cluster info of the k-means algorithm.
- *Dispersion Matrix:* this last panel shows the matrix of all the possible variable vs. variable scatter plots (Figure 7.46). See the comments reported above for the dispersion matrix of the k-means algorithm.

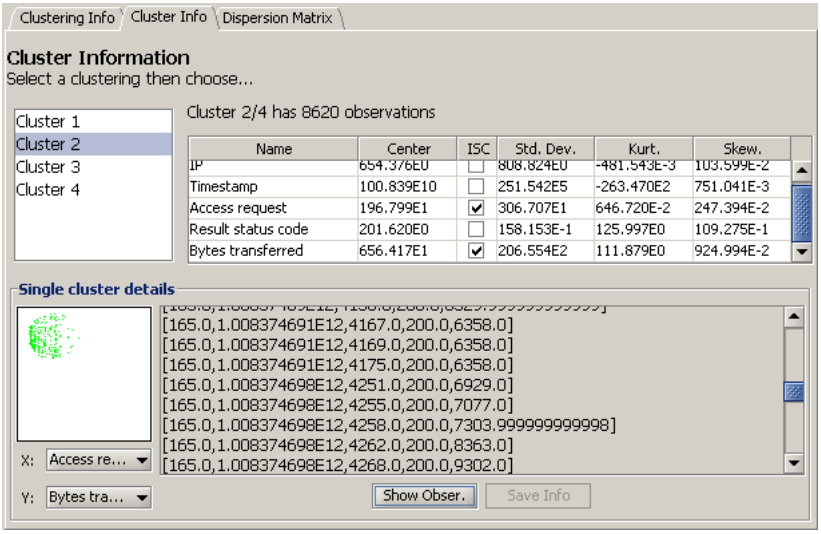


Figure 7.45: Descriptive statistics of the clusters obtained with the fuzzy k-means algorithm and their visual characterization through the dispersion matrix

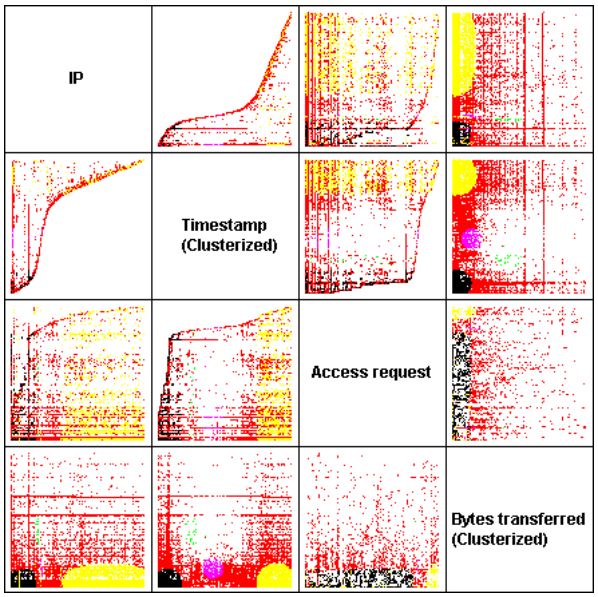


Figure 7.46: Fuzzy k-Means scatter plot matrix. Each observation is represented with the color corresponding to the cluster it is assigned

7.1.7 An example of a web log analysis

In this section we will describe an example of application of the JWAT tool for a workload analysis. The analyzed set of data consists of observations stored in the log file of a web server of a university site.
The input file *Demo1Data.jwat* and its format *Demo1Format.jwatformat* can be found in the subfolders *Examples* and *jwatFormats*, respectively, of the Java Modelling Tools folder installed with the JMT tools.

Step 1 - Input panel

```
146.48.13.191 15/Dec/2001:00:00:04 "GET /homepage/homepage.php" 3914
146.48.13.191 15/Dec/2001:00:00:05 "GET /homepage/homepage.php" 7825
151.29.12.105 15/Dec/2001:00:00:06 "GET /images2000/home/ombrasmussoliv2.gif" 223
151.29.12.105 15/Dec/2001:00:00:06 "GET /images2000/home/ombraliv2.gif" 131
151.29.12.105 15/Dec/2001:00:00:06 "GET /images2000/home/spacer.gif" 43
```

Figure 7.47: Example of observations in the input file considered.

Name	Type	Select	Comment	Sep.	Perl5 Reg. Exp.	Def.	Rep.	
IP	String	☑	IP Address		\d+.\d+.\d+.\d+			✖
Timestamp	Date	☑	Timestamp of the visit as s...		\d\d/\w\w\w/\d\d...			✖
Access request	String	☑	The request made	"	.+			✖
Bytes transferred	Numeric	☑	The number of bytes trans...		([+-])?\d+([.]\d+)?			✖

Figure 7.48: Input format example

Each observation consists of the values of 4 variables described in Figure 7.48:

- *IP*: String type variable that defines IP address of the request.
- *Timestamp*: Date type variable that defines the timestamp of the request, the regular expression

  `"\d\d/\w\w\w/\d\d\d\d:\d\d:\d\d:\d\d"'`

  identifies that the data will be in the format dd/mmm/yyyy:hh:mm:ss (e.g., 12/Jun/2000:12:21:45).
- *Access request*: String type variable that identifies the object that is the target of the request. The definition through delimiter " and regular expression "'.*"' allows the reading of all the string that is contained between quotation marks; if it is necessary it is possible to modify the expression "'.*"' to read only a part of the string between quotation marks.
- *Bytes transferred*: Numeric type variable that identifies the quantity of bytes transferred due to the execution of the request submitted.

Step 2 - Descriptive statistics and sample extraction
Before the execution of the cluster analysis the size of the input data set has been reduced executing the filter *Interval* to the values of the variable *Bytes transferred* setting a maximum value of 20000 and a minimum value of 0. The purpose of this filtering operation is to obtain a data set more compact but still representative of the original one. In Tables Figure 7.49 and Figure 7.50 the descriptive statistics before and after applying the filter on the input data are shown, respectively. Scatter plot matrix of the filtered data are shown in Figure 7.51.

Variable	Mean	Variance	Std. Dev.	Coeff. of v...	Minimum	Maximum	Range	Median	Kurtosis	Skewness	Num. Obs.
IP	516.057E0	144.585E3	380.244E0	736.825E-3	000.000E0	124.800E1	124.800E1	473.000E0	-123.437E-2	312.164E-3	376.140E2
Timestamp	100.840E10	271.278E12	164.705E5	163.333E-7	100.837E10	100.842E10	477.690E5	100.841E10	-123.961E-2	-642.748E-3	376.140E2
Access request	384.799E0	312.824E3	559.306E0	145.350E-2	000.000E0	232.000E1	232.000E1	131.000E0	212.949E-2	182.649E-2	376.140E2
Bytes transferred	397.872E1	465.577E5	682.332E1	171.495E-2	000.000E0	398.790E2	398.790E2	111.100E1	598.435E-2	255.522E-2	376.140E2

Figure 7.49: Descriptive statistics of the original data before applying the filter on the "Bytes transferred" variable

About 2500 observations have been dropped due to the filtering action, reducing considerably the variance.

Step 3 - Clustering panel
The k-Means algorithm and the variables *timestamp* and *Bytes transferred* have been selected in the clustering panel together with the following options: number of clusters 7, interactions 50 and transformation *Max-Min*. The clustering results panel is opened automatically. Through the *'Back'* button we went back to the previous panel to execute a new clustering algorithm. The fuzzy k-Means algorithm and the variables *timestamp* and *Bytes transferred* have been selected together with the following options: number of clusters 6, interactions 20, fuzziness level 2.0.

Variable	Mean	Variance	Std. Dev.	Coeff. of v...	Minimum	Maximum	Range	Median	Kurtosis	Skewness	Num. Obs.
IP	513.059E0	144.963E3	380.740E0	742.098E-3	000.000E0	124.600E1	124.600E1	466.000E0	-123.152E-2	326.854E-3	351.270E2
Timestamp	100.840E10	271.088E12	164.647E5	163.276E-7	100.837E10	100.842E10	477.690E5	100.841E10	-125.431E-2	-628.080E-3	351.270E2
Access request	389.184E0	312.667E3	559.166E0	143.677E-2	000.000E0	232.000E1	232.000E1	141.000E0	201.294E-2	179.578E-2	351.270E2
Bytes transferred	241.119E1	117.579E5	342.898E1	142.211E-2	000.000E0	199.800E2	199.800E2	100.200E1	610.155E-2	237.149E-2	351.270E2

Figure 7.50: Descriptive statistics of the original data after applying the filter on the "Bytes transferred" variable

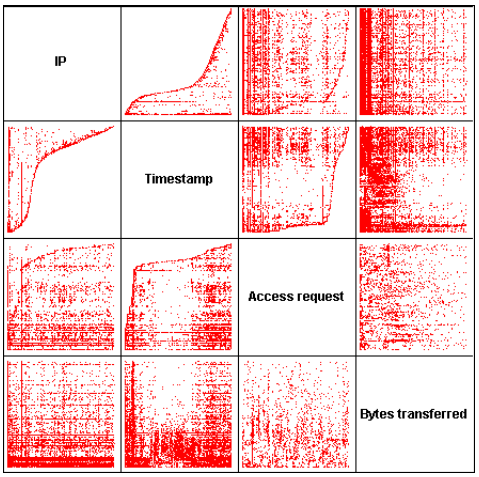


Figure 7.51: Scatters matrix on the data set after applying the filter on the "Bytes transferred" variable.

Step 4 - Results panel

The results concerning the goodness of the partitions obtained with the two clustering algorithms are reported in the following tables:

Clustering	Cl.
k-Means	7
Fuzzy k-Means	6

Clustering algorithms

Cl	G	OMSR	Rat
2	✓	876.308E2	893.69
3	✗	980.541E1	564.59
4	✗	173.673E2	114.89
5	✓	151.157E2	238.12
6	✗	634.773E1	961.66
7	✗	660.077E1	000.00

k-Means

Cl	Entropy	Ratio
2	200.614E-3	-
3	433.346E-3	216.009E-2
4	552.932E-3	127.596E-2
5	651.680E-3	117.859E-2
6	674.182E-3	103.453E-2

Fuzzy k-Means

We have visualized different statistics for each algorithm by varying the number of clusters, in particular we show the scatter plots matrix for the partition with 5 clusters (as can be seen from the OMSR values this partition has been evaluated as a good one).

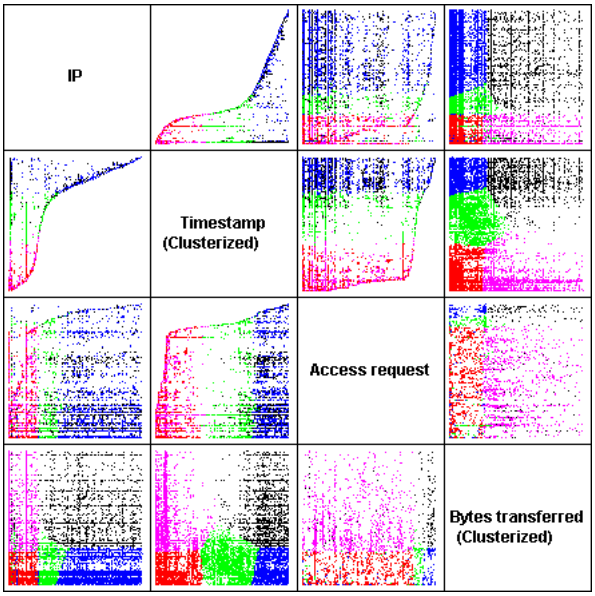


Figure 7.52: K-Means algorithm scatter plots matrix

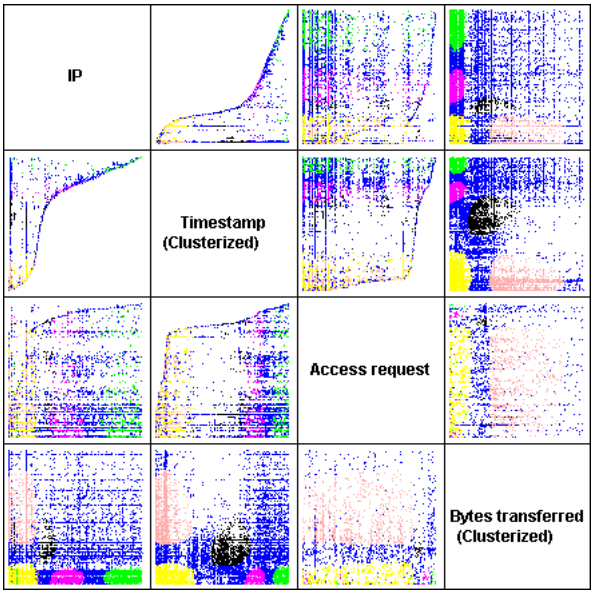


Figure 7.53: Fuzzy k-Means algorithm scatter plots matrix

7.1.8 Menus description

File

New Use this command to start a new analysis selecting a new log file.

Shortcut on Toolbar: 
Accelerator Key: CTRL+N

Exit

Use this command to terminate the JWAT current session. It is possible to utilize '`Exit`' button in the buttons bar of application. If the session has been modified after its creation or after its last saving, a popup window asks confirmation to save session.

Accelerator Key: CTRL+Q

Action

Solve Use this command, when enabled, to start the algorithm selected in the clustering panel. It is possible to obtain the same result by using '`Solve`' button in the buttons bar of the application.

Shortcut on Toolbar: 
Accelerator Key: CTRL+L

Help

JWAT Help Use this command to visualize the help. Suggestions concerning the single tab in the application are also available by using the '`Help`' button in the buttons bar of application.

Shortcut on Toolbar: 
Accelerator Key: CTRL+Q

About

Use this command to visualize information concerning JWAT application.

7.1.9 Demo

In order to show the functionality of the workload analysis program, a demo has been realized. It can be accessed using the button in the main JWAT panel (see Figure 7.1). Once the demo panel has been opened it is sufficient to click on `Load demo` button (see Figure Figure 7.54) to launch the workload analysis algorithms on the file `Demo_WAData.jwat` present in the folder `examples` of the JMT installation directory.

7.2 Fitting

To start a fitting session press the button  in the JWAT window and the window of Figure 7.55 will appear. This application helps the user to assess whether a sequence of experimental data fits a known distribution through the application of statistical hypothesis algorithms. In case the experimental data fits one of the available known distributions the parameter values to be used to reproduce the original sequence of data will be shown. Graphs will also be produced. With the current JWAT release only the Pareto and exponential distributions are available, more will be added in the future. The main components of the fitting window are:

- *Menu and Toolbar*: the functioning of these two parts of the window is the same of the one of the Workload Analysis application, so the reader is invited to see the introduction of Section 7.1.
- *Tabbed pane*: It concerns the main functions of the application. All the operations and results obtained during the fitting procedure are shown in three navigable tabs: input, Pareto and exponential (see Figure 7.56).
- *Navigation buttons*: this part aims at providing an alternative method of navigation though the tabbed pane (other than clicking on the tab themselves), using the next and previous buttons. For a detailed description see Section 7.1.

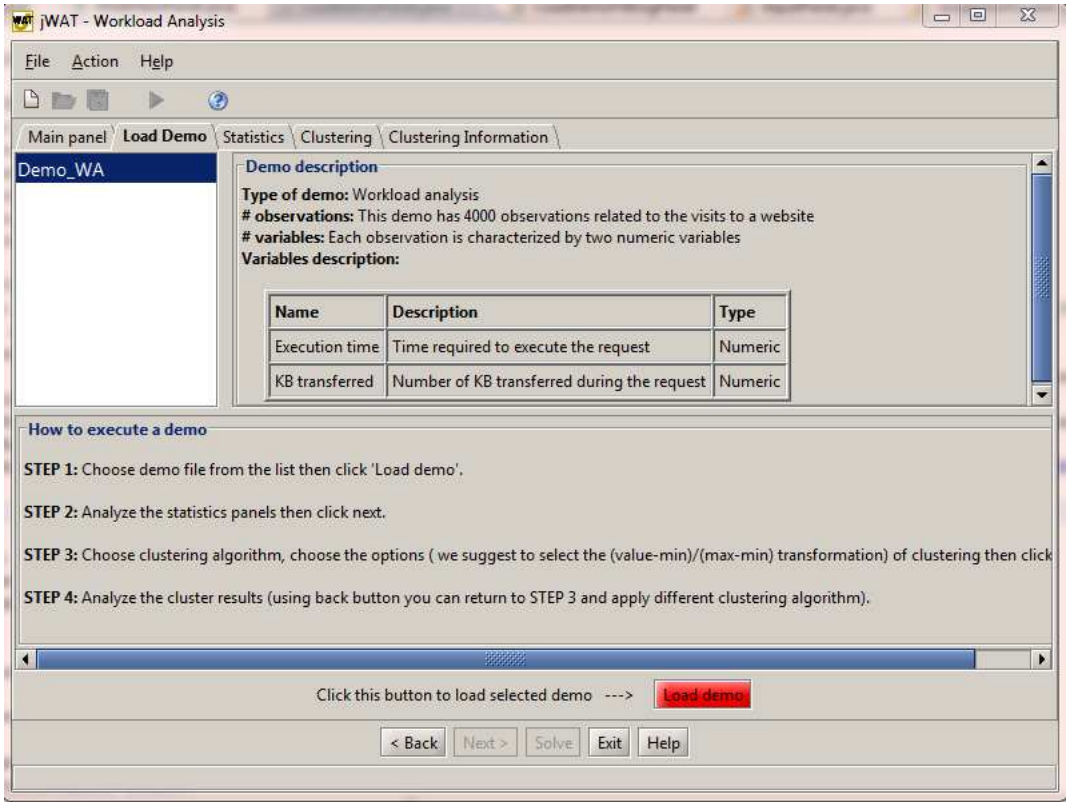


Figure 7.54: Demo Panel of the workload analysis tool

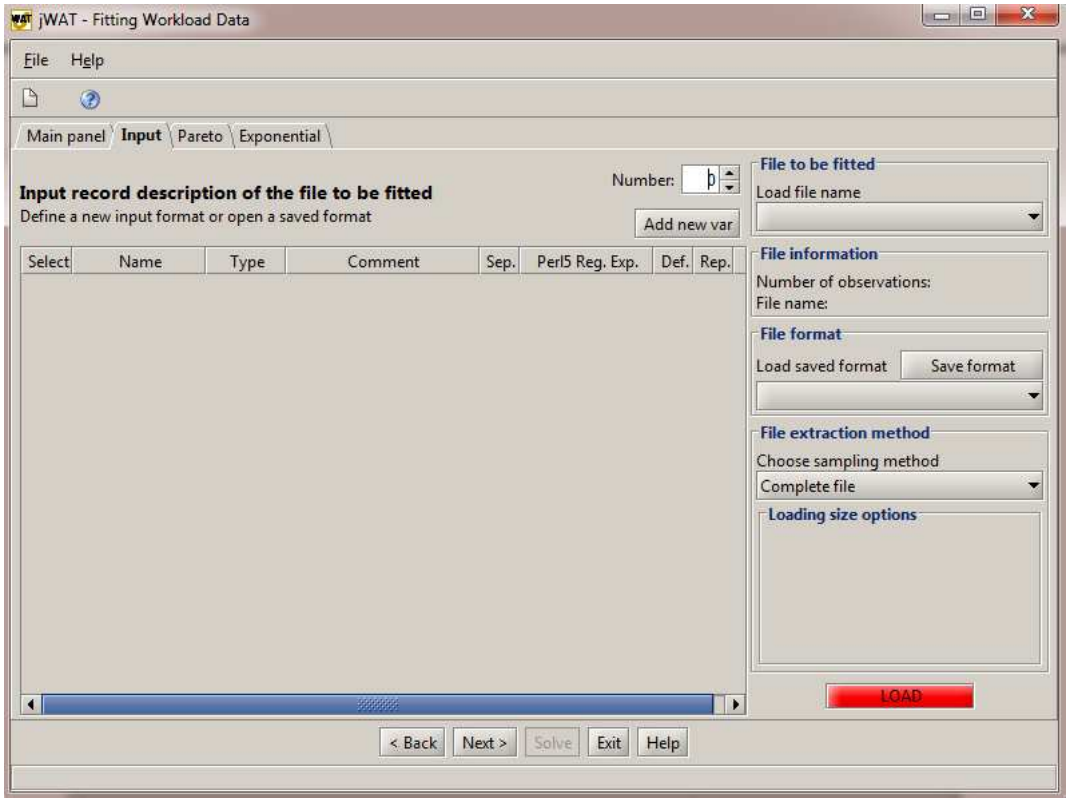


Figure 7.55: Main window of the Fitting application

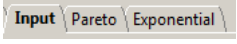


Figure 7.56: Tabs selector

In the following sections the utilization of the panels of the fitting application will be described in detail.

7.2.1 Input definition

The definition of the input is similar to the one of Workload Analysis described in Section 7.1.2. In this step it is necessary to specify only one numeric variable in order to apply the fitting function. If more than one numeric variable should be fitted it is necessary to reload each time the log specifying a different Numeric field.

7.2.2 Pareto and Exponential panels

The Pareto and Exponential panels show the result of the application of the fitting function to the data provided in the Input definition, together with some graphical representations of statistical functions. The operation performed by the application can be subdivided into three steps, that are performed automatically:

- *Application of hypothesis tests*: two goodness-of-fit tests are applied to the data. In particular, the Pareto distribution is tested using the algorithm described in [**?**] while the exponential distribution using the one in [**?**]. Note that these tests have a good statistical power function in the presence of a consistent number of observations, thus for a reliable result use at least 50 or 100 observations for both tests.
- *Estimation of the parameters*: if a test gives a positive outcome then the parameters are estimated using a Maximum Likelihood Estimator. The estimators of the scale (k) and shape (α) parameters for the Pareto distribution are provided in [**?**] while the unique parameter λ of the exponential is simply the inverse of the empirical mean.
- *Construction of the graphs*: this step builds three graphs for each of the two panels: a QQ-Plot, an empirical Cumulative Function (CDF) and an empirical density function (pdf). The graphs are useful to make a visual assessment of the fitting of the data to the particular distribution, beyond the result of the goodness-of-fit test itself. Each of the graphs can be enlarged double clicking on it (see Figure 7.59). It is also possible to zoom-in and zoom-out the graphs, and also to save the plot in both `png` and `eps` formats.

Let us make an example of the behavior of the fitting application analyzing a sequence of data that will result to follow a Pareto distribution. The data associated with this example can be found in the directory `example` under the name *pareto_ example.jwat* and loading the format *pareto_ format.jwat*.

The Pareto panel in Figure 7.57 shows the results of the fitting with respect to the Pareto distribution. The upper part of the panel contains the message that the test applied with a significativity of 0.05 produced a positive outcome. After that, the estimation of the parameters of the distribution are provided, in particular α (shape parameter) and k (scale parameter).

Finally, three typical statistical indicators are showed: the mean μ, the variance σ^2 and the coefficient of variation σ/μ. The lower part of the panel contains three graphs. The first one on the left is the QQ-Plot, which displays the comparison between the data (x-axis) and the quantiles of the Pareto distribution having the estimated parameters. The 45 degree line represents a reference for the points of the QQ-Plot, and it's the place where the data would lie if they followed perfectly a distribution with the estimated parameters. If the data are Pareto distributed, the points should lie close to the continuous red line. The graph in the middle is the CDF of the analyzed sequence of data, and it is constructed incrementing the value of the y-axis of $1/n$ every time a point of the dataset is encountered. The last graph is the empirical pdf of the analyzed data, which is essentially a frequency graph obtained dividing the x-axis in a number of buckets proportional to number n of observations, and counting the percentage of points that falls in each bucket.

Figure 7.58 shows the results of the the exponential fitting on the same sequence of data. As it can be seen, the message says now that the data are not exponentially distributed. The mean, variance and coefficient of variation of the analyzed data are showed. The QQ-Plot in the lower-left part confirms that the data is not exponentially distributed, since the majority of the points lie far away from the 45 degree line.

7.2.3 Demo

In order to show the functionality of the fitting program, a demo has been realized. It can be accessed using the button in the main JWAT panel (see Figure 7.1). Once the demo panel has been opened it is sufficient to click on `Load demo` button (see Figure Figure 7.60) to read and fit automatically the file `DemoFittingPareto.jwat` present in the folder `examples` of the JMT installation directory. As the name suggests, this demo shows the fitting procedure applied to a dataset that is Pareto distributed.

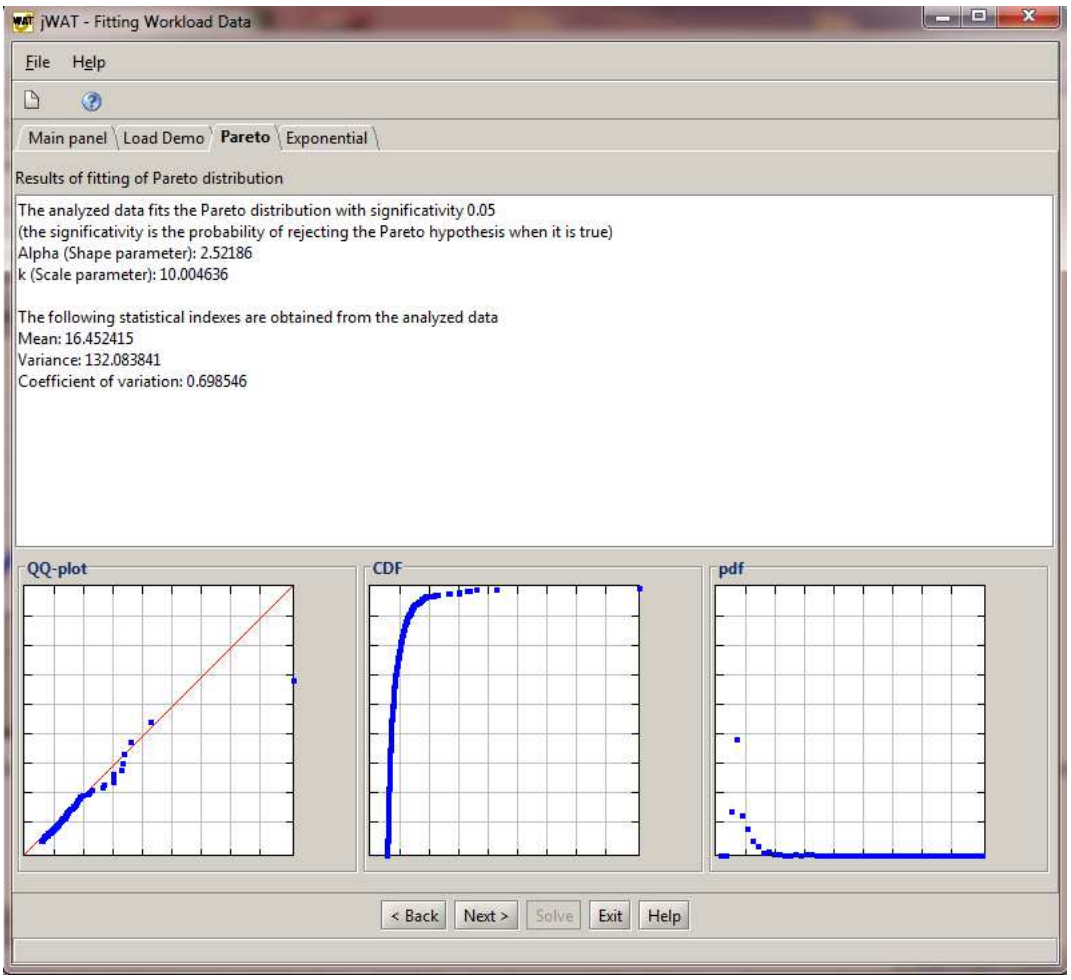


Figure 7.57: Results of the fitting on the Pareto panel when the program is fed with Pareto-distributed data

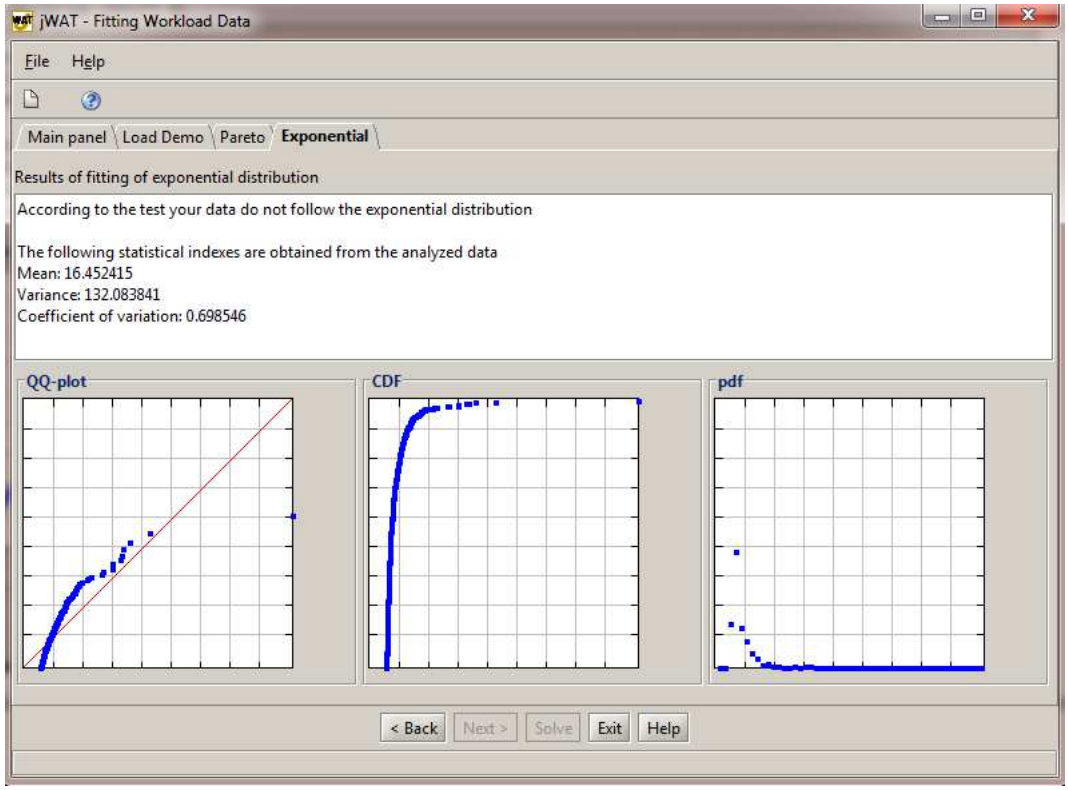


Figure 7.58: Results of the fitting on the exponential panel when the program is fed with Pareto-distributed data

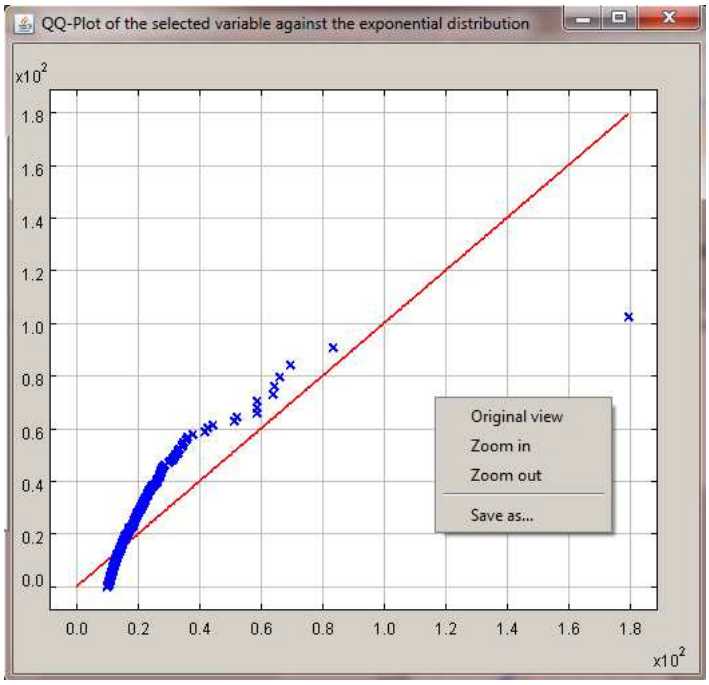


Figure 7.59: Enlarged QQ-Plot

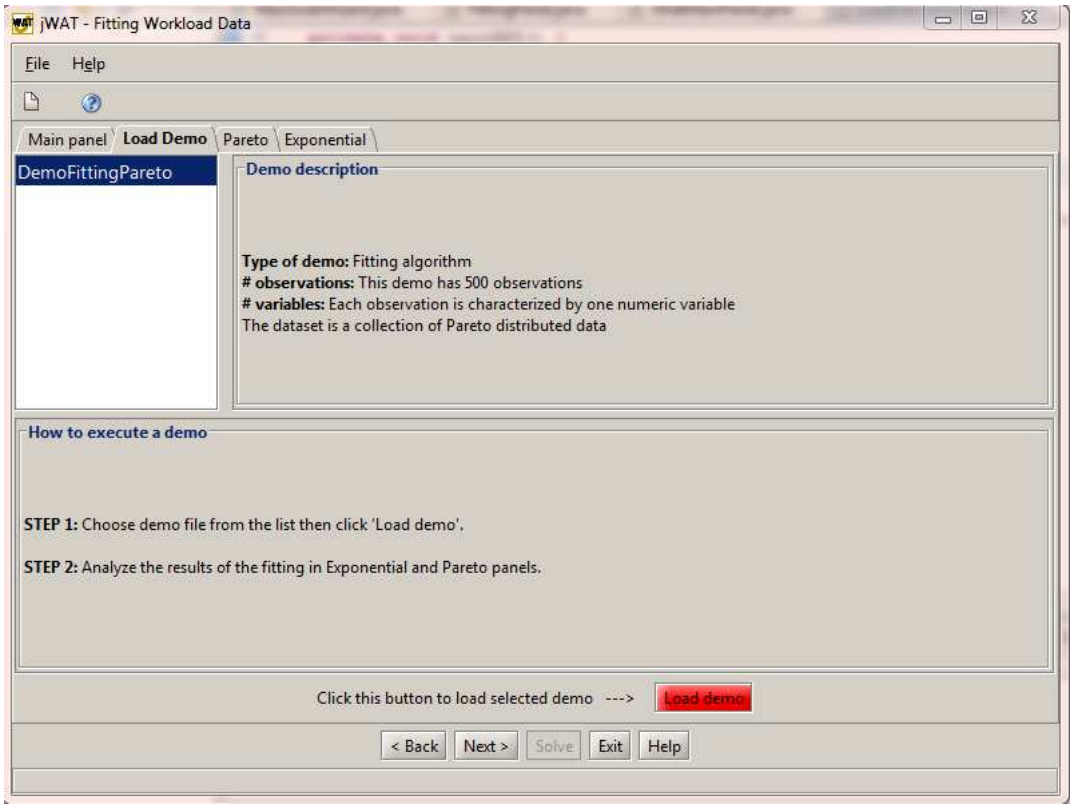


Figure 7.60: Demo Panel of the fitting tool

Appendix A

Basic Definitions

Approximate Mean Value Analysis (AMVA). Algorithms for computing approximate solutions of multi-class closed queuing networks providing significant time and space reduction by substituting non-recursive approximations for queue lengths (per resource r and per class c) seen by an arriving customer $A_{r,c}(\vec{N})$.

Arrival Rate (λ). Rate at which customers of an open class arrive to the system.

Batch workload. Closed workload class composed of customers that do not require interactions with users during their execution (i.e., the users think time is zero).

Bottleneck Station. The station with the highest utilization in the system.

Class of customers. A group of customers with service demands on the different stations statistically equal. Open classes are specified by the *arrival rate* λ (job/s), closed classes consist of a constant number of jobs N specified as the *population size* (job).

Class Switch. A *class switch* station allows the customers to change class according to the probabilities described in a *class switch matrix*

Closed Class. Workload class in which customers that have completed service leave the system and are instantaneously replaced by a new customer. A closed model has a fixed population of requests.

Customer or **Job** or **Transaction** or **Request**. It is the element that will require service to stations. For example, it can be an http request, a database query, a ftp download command, or an I/O request.

Delay station. Customers at a delay station are each (logically) allocated their own server, so there is no competition for service (no queue). The time spent by a customer at a delay station is exactly its service demand. In delay stations, for consistency with Little's law, the *utilization* is computed as the *average number of customers* in the station, and thus it may be *greater than 1*

Dominated Stations. A station S is dominated if exist at least one station Q whose service demands for each class of customer are highest or equal (but at least one highest) than those of S.

Interactive workload (terminal workload). Its intensity is specified by two parameters: the number of active terminals (customers), and the average length of time that customers use terminals ("think time") between interactions.

Load Dependent Resource (station). A load dependent service station can be thought of as a service station whose service rate (the reciprocal of its service time) is a function of the customer population in its queue.

Load Independent Resource (station). A load independent service station is a station whose service time S is *independent* of the actual load of the station (i.e., of the number of customers in the station).

Masked-off Stations. a station that is not a Potential Bottleneck or a Dominated station. A Masked-off station may exhibit the largest queue-length and hence the highest response time.

Mean Value Analysis (MVA). Iterative technique used to evaluate closed queuing networks.

Multiple class models. Models with several customer classes, each of which has its own workload intensity and its own service demand at each center. Within each class, the customers are indistinguishable.

Number of customers (N_r) . Average number of customers at station r, either waiting in queue *and* receiving service.

Number of customers in the system. It is the aggregate measure of the *Number of customers* over all the stations.

Number of visits. The average number of visits that a customer makes to each resource during a complete execution.

Open Class or **Transaction Workload**. Workload class in which there is an external source with an arbitrary number of customers that are sent to the system according to a given distribution. The number of customers in the model varies over time and can be arbitrarily large depending on the congestion level of the resources and on the arrival process.

Global Population (N). Constant number of requests belonging to a closed class.

Population Mix (β_i). The ratio between N_i, the *number of customers* of closed class i, and the total number of customers in the system: $\beta_i = N_i / \sum_k N_k$. It can be defined either for open and closed models.

Potential Bottleneck Station. A station that can become bottleneck for some feasible population mixes.

Queueing network model. A network of queues is a collection of service resources, which represent system resources, and customers, which represent users or transactions.

Queueing station. A resource consisting of two components: queue and server. Customers at a queueing resource compete for the use of the server. The time spent by a customer at a queueing resource has *two* components: time spent waiting in queue and time spent receiving service.

Reference Station. User-specified station where a customer flow through when it has completed its execution, i.e., it has performed an entire cycle of service in the system. The system throughput, system response time, and the visits V_k are computed with respect to the visits at the reference station (*usually* assumed as 1).

Residence Time (W_k). Average time that a customer spent at station k during its complete execution. It includes time spent queueing and time spent receiving service. It does not correspond to *Response Time* R_k of a station since $W_k = R_k * V_k$.

Response Time (R_k). Average time required by a customer to flow through a station k (includes queue time and service time).

Response Time per Sink. average time spent in system by a customer before dropping at the selected sink.

Saturated Resource. When the arrival rate at a resource is greater than or equal to the maximum service rate, the resource is said to be saturated (i. e., its utilization is equal to 1).

Saturation Sector. A set of Population Mixes (i.e., fraction of customers of the different classes that are in execution) in which *two or more* stations are *saturated concurrently*, i.e., are *bottlenecks*.

Server Utilization (U). The utilization of a queuing center is the proportion of time the resource is busy or, equivalently, the average number of customers in service there (definition valid also for delay centers).

Service Demand (D_k). average service requirement of a customer, that is the total amount of service required by a complete execution at resource k. In the model it is necessary to provide separate service demand for each pair service center-class. It is given by $D_k = V_k * S_k$.

Service Time (S_k). Average service requirement of a request of a given class per visit at resource k.

Station or **Service Center** or **Resource**. It represents an element of the network. Customers arrive at the station and then, if necessary, wait in queue, receive service from server, and depart from the station.

System Power (ϕ). It is a performance metric that allows to take into consideration *both* the system throughput X and the system response time R [Kle79]. It is defined as X/R. A system is in the optimal operational point when the value of the power is maximized. For a station c of an open model consisting of a single queue station that works in the optimal operational point it will be: $\lambda_c = 0.5\, S,\ U_c = 0.5, N_c = 1$.

System Response Time (R). Correspond to the intuitive notion of response time perceived by users, that is, the time interval between the instant of the submission of a request to a system and the instant the corresponding reply arrives completely at the user. It is the aggregate measure of *Residence Times*: $R = \sum_k W_k$.

System Throughput (X). Rate at which customers perform an entire interaction with the system (i.e., a complete execution). It is the throughput observed at a user-defined *reference station*.

Throughput (X_r). Rate at which customers are executed by station r accounting also periods where the server is idle. That is, X_r is the mean number of completions in a time unit from station$r$.

Throughput per Sink. rate at which customers departs from the system with respect to the selected sink, i.e., the number of requests completed in a time unit that reach a given sink.

Utilization (U_r). Proportion of time in which the station r is busy or, in the case of a *delay* station, it may be interpreted as the average number of customers in the station (see Little Law [Lit61]).

Visit (number of -) (V_k). Average number of visits that a customer makes at station k during a complete execution; it is computed as the ratio of the number of completions at resource k to the number of system completions as observed at a user-defined reference station. If resource k is a delay center representing the users, usually it is assigned a unitary value to the number of visits to this station.

Appendix B

List of Symbols

λ_c arrival rate of class c
N global population of a model
β_c population mix $\beta_c = N_c / \sum_k N_k$, fraction of class-c customers
N_k number of customers at station k
V_k average number of visits at station k
W_k average residence time at station k, $W_k = R_k V_k$
R_k response time of station k
U_k utilization of station k , $U_k = X_k S_k = X_0 V_k S_k = X_0 D_k$
D_k service demand at station k (single class)
$D_{k,c}$ service demand at station k of class c customer
R response time of the system, $R = \sum_k W_k$
X system throughput
X_k throughput at station k
ϕ system power ($\phi = X/R$)

Appendix C

Acknowledgement

The codebase and/or manual of JMT includes contributions from the following people:

- Politecnico di Milano (Coordinator: Giuseppe Serazzi): Marco Bertoli, Emma Bortone, Davide Brambilla, Arif Canakoglu, Mattia Cazzoli, Davide Cerotti, Andrea Conti, Federico Dall'Orso, Francesco D'Aquino, Ashanka Das, Ernesto Di Mauro, Claudio Fumagalli, Federico Granata, Carlo Gimondi, Marco Gribaudo, Stefano Omini, Francesco Radaelli, Kourosh Sheikhvand, Sebastiano Spicuglia, Andrea Zanzottera.
- Imperial College London (Coordinator: Giuliano Casale): John Bradshaw, Abhimanyu Chugh, Emyr Davies, Michalis Makaronidis, Anastasia Eleftheriou, Shuai Jiang, Vitor Lopes, Srikrishna Murali, Rosemary Lok Sen Ng, Phumin Phuangjaisri, Ahmed Salem, Haixiao Su, Piotr Tokaj, Krishnakumar Vaibhav, Hong Ong Wai, Alexander Zakon, Lulai Zhu

Bibliography

[Bau93] F. Bause. Queueing petri nets: a formalism for the combined qualitative and quantitative analysis of systems. In *Proc. of the 5th Int.l Workshop on Petri Nets and Performance Models*, pages 14–23, Toulouse (France), 1993. IEEE Press.

[BBC+81] G. Balbo, S.C. Bruell, L. Cerchio, D. Chialberto, and L. Molinatti. *Mean value analysis of closed load dependent queueing networks*. Dipartimento di Informatica, Universita' di Torino, Oct.1981.

[BC84] J. Banks and J. Carson. *Discrete-Event System Simulation*. Prentice-Hall, 1984.

[BCMP75] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, closed and mixed networks of queues with different classes of customers. *J.ACM*, 22:248–260, 1975.

[BCS06] M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proceedings of QEST 2006 Conference*, pages 119–120, Riverside, US, Sep 2006. IEEE Press.

[BCS07] M. Bertoli, G. Casale, and G. Serazzi. The jmt simulator for performance evaluation of non-product-form queueing networks. In *Simulation Symposium, 2007. ANSS '07. 40th Annual*, pages 3–10, March 2007.

[BK02] F. Bause and P.S. Kritzinger. *Stochastic Petri Nets: An Introduction to the Theory*. Vieweg Verlag, 2002.

[BKK84] O.J. Boxma, F.P. Kelly, and A.G. Konheim. The product form for sojourn time distributions in cyclic exponential queues. *Journal of the ACM (JACM)*, 31(1):128–133, 1984.

[BKLC84] Raymond M Bryant, Anthony E Krzesinski, M Seetha Lakshmi, and K Mani Chandy. The mva priority approximation. *ACM Transactions on Computer Systems (TOCS)*, 2(4):335–359, 1984.

[BKT83] Raymond M. Bryant, Anthony E. Krzesinski, and Peter Teunissen. The mva pre-empt resume priority approximation. In *Proceedings of the 1983 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '83, page 12–27, New York, NY, USA, 1983. Association for Computing Machinery.

[BS96] G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks: Common bottlenecks. *Performance Evaluation*, 26(1):51–72, 1996.

[BS97] G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks: Multiple bottlenecks. *Performance Evaluation*, 30(1):115–152, 1997.

[Cas06] G. Casale. An efficient algorithm for the exact analysis of multiclass queueing networks with large population sizes. *SIGMETRICS/Performance*, pages 169–180, 2006.

[Cas09] G. Casale. Comom: Efficient class-oriented evaluation of multiclass performance models. *IEEE Transactions on Software Engineering*, 35:162–177, 2009.

[Cas11] G. Casale. Exact analysis of performance models by the method of moments. *Perf. Evaluation*, 68(6):487–506, 2011.

[CFGG95] G. Chiola, G. Franceschinis, R. Gaeta, and M. Gribaudo. Greatspn 1.7: Graphical editor and analyzer for timed and stochastic petri nets. *Performance Evaluation*, 24(1-2):47–68, 1995.

[CG86] A.E. Conway and N.D. Georganas. Recal—a new efficient algorithm for the exact analysis of multiple- chain closed queuing networks. J.ACM, Vol 33, pp.768-791, 1986.

[Cho83] W.M. Chow. Approximations for large scale closed queueing networks. *Performance Evaluation*, 3(1):1–12, 1983.

[CL83] S. Chandy and M. S Lakshmi. An approximation technique for queueing networks with preemptive priority queues, February 1983.

[CN82] K. M. Chandi and D. Neuse. Linearizer: a heuristic algorithm for queueing network models of computing systems. *Comm. of the ACM*, 25(2):126–134, 1982.

[CS04] G. Casale and G. Serazzi. Bottlenecks identification in multiclass queueing networks using convex polytopes. In *Proc. of IEEE MASCOTS Symposium*, pages 223–230. IEEE Press, 2004.

[CS11] G. Casale and G. Serazzi. Quantitative system evaluation with java modelling tools. In *Proc. ICPE11: ACM-SPEC Int.l Conf. on Performance Engineering*, pages 449–454. ACM, 2011.

[CZS07] G. Casale, E.Z. Zhang, and E. Smirni. Characterization of moments and autocorrelation in maps. *ACM Perf. Eval. Rev., Special Issue on MAMA Workshop*, 35(1):27–29, 2007.

[DKS09] N.J. Dingle, W.J. Knottenbelt, and T. Suto. Pipe2: A tool for the performance evaluation of generalised stochastic petri nets. *SIGMETRICS Perform. Eval. Rev.*, 36(4):34–39, 2009.

[dSeSM90] E. de Souza e Silva and R.R. Muntz. A note on the computational cost of the linearizer algorithm for queueing networks. *IEEE Transactions on Computers*, 39(6):840–842, 1990.

[Fis73] G. S. Fishman. Statistical analysis for queueing simulations. *Management Science, Series A (Theory)*, 20(3):363–369, 1973.

[GAM78] A. V. Gafarian, C. J. Ancker, and T. Morisaku. Evaluation of commonly used rules for detecting "steady state" in computer simulation. *Naval Research Logistics Quarterly*, 25:511–530, 1978.

[HL04] Peter G. Harrison and Ting Ting Lee. A new recursive algorithm for computing generating functions in closed multi-class queueing networks. *Proc. of the 2004 IEEE Int.l MASCOTS Symposium*, pages 223 – 230, 2004.

[HW81] P. Heidelberger and P.D. Welch. A spectral method for confidence interval generation and run length control in simulations. *Commun. ACM*, 24(4):233–245, 1981.

[JD82] J.McKenna and D.Mitra. Integral representations and asymptotic expansions for closed markovian queueing networks: Normal usage. *Bell Systems Tech*, 61:661 – 683, 1982.

[J.P73] J.P.Buzen. Computational algorithms for closed queueing networks with exponential servers. *Journal of the Association for Computing Machinery*, 16(10):527–531, 1973.

[KDB06] S. Kounev, C. Dutz, and A. Buchmann. Qpme - queueing petri net modeling environment. In *Proc. 3rd Int.l Conference on Quantitative Evaluation of SysTems (QEST-2006)*, Riverside, US, 2006.

[Kle75] L. Kleinrock. *Queueing Systems*. Wiley Interscience, 1975.

[Kle79] L. Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. *Proc. International Conference on Communication*, pages 43.1.1–43.1.10, June 1979.

[Lit61] J. D. C. Little. A proof of the queueing formula l= λw. *Operations Research*, 9:383–387, 1961.

[LZGS84] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[MBC+95] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalised Stochastic Petri Nets*. John Wiley and Sons, 1995.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.

[Neu89] M.F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, 1989.

[oTCSRGS77] University of Toronto. Computer Systems Research Group and KC Sevcik. Priority scheduling disciplines in queueing network models of computer systems. In *IFIP Congress*, 1977.

[Paw90] K. Pawlikowski. Steady-sate simulation of queueing processes: A survey of problems and solutions. *ACM Computing Surveys*, 2(22):123–168, 1990.

[Pet62] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universitat Bonn, 1962.

[RK76] M. Reiser and H. Kobayashi. On the convolution algorithm for separable queuing networks. *Proc. of the 1976 ACM SIGMETRICS Conf. on Computer performance modeling measurement and evaluation*, pages 109–117, 1976.

[RL80] M. Reiser and S.S. Lavenberg. Mean-value analysis of closed multichain queuing networks. J.ACM, Vol 27, No 2, pp.313-322, April 1980.

[Rob05] S. Robinson. Automated analysis of simulation output data. In *In Proc. Winter Sim. Conf. 2005*, pages 763–770, 2005.

[RW97] K.W. Ross and J. Wang. Implementation of monte carlo integration for the analysis of product-form queueing networks. *Performance Evaluation*, 29:273 – 292, 1997.

[Sch79] P. J. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proceedings of Int.l Conf. on Stochastic Control and Optimization*, pages 25–29, 1979.

[Sch82] L. Schruben. Detecting initialization bias in simulation output. *Operations Research*, 30:569–590, 1982.

[Spr98] S.C. Spratt. *Heuristics for the startup problem*. Unpublished M.S. Thesis, Department of Systems Engineering, University of Virginia, 1998.

[Ste09] W.J. Stewart. *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, 2009.

[TGBB17] Ijjou Tizgui, Fatima Guezar, Hassane Bouzahir, and Brahim Benaid. Comparison of methods in estimating weibull parameters for wind energy applications. *International Journal of Energy Sector Management*, 11, 09 2017.

[Tri02] K.S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Wiley, 2002.

[TS85] Salvatore Tucci and Charles H. Sauer. The tree mva algorithm. *Performance Evaluation*, 5(3):187–196, 1985.

[WCS00] Jr.K.P. White, M. J. Cobb, and S. C. Spratt. A comparison of five steadystate truncation heuristics for simulation. pages 755–760, 2000.

[WP78] J. R. Wilson and A. A. B. Pritsker. A survey of research on the simulation startup problem. *Simulation*, 31(2):55–58, 1978.

[ZES88] J. Zahorjan, D.L. Eager, and H.M. Sweillam. Accuracy, speed, and convergence of approximate mean value analysis. *Performance Evaluation*, 8(4):255–270, 1988.

Index

Advanced Metric Configuration, 58
Algorithm
    AMVA, 5, 18
    AQL, 5, 20
    Bard-Schweitzer, 5, 20
    Chow, 5, 18
    CoMoM, 5
    De Souza-Muntz, 5, 20
    Linearizer, 5, 20
    MoM, 5
    MVA, 5
    RECAL, 5
    tolerance, 18
    TreeMVA, 5
AMVA, 5, 18
    stopping criterion, 18
    tolerance, 18
Animation interval
    simulation results, 106
Approximate algorithms, 18
AQL algorithm, 20
arc connections, 42
Arrival process, 39, 139
Arrival Rate, 61
Arrival rate, 55
Asymptotic technique, 179
Automatic connections, 41

Balking Rate, 55
Bard-Schweitzer algorithm, 20
BAS blocking, 163
BAS scheduling, 39, 100, 163
BCMP theorem, 69, 105, 149
Bezier curves, 42
Bottlenecks
    characteristic polytope, 185
    convex hull, 184
    dominated, 185
    masked-off, 185
    natural bottlenecks, 183
    polyhedral analysis, 179, 185
    potential bottleneck, 185
    saturation sector, 183
    switching points, 184
    switching segments, 184
    system bottlenecks, 183, 184
    with multiclass workload, 179

Capacity, 69
Chow algorithm, 18
Class Switch
    routing, 68
Class switch, 146
    and model equilibrium, 90
    example, 88, 123
    implementation details, 90
    limitations, 90
    matrix, 146
    probability matrix, 88
    station, 88
Classes of customers, 45
    Closed, 47
    Open, 46
Closed model
    Number of Customers, 45
Clustering, 210
    Fuzzy k-means, 212, 216
    K-means, 212, 213
Coefficient of variation, 48
Command line
    JMVA, 33
    JSIM*graph*, 124
    XML of a model, 36
    XML of results, 37
Confidence intervals, 58, 108, 163
CPNs, 92
Cycle time, 11

De Souza-Muntz algorithm, 20
Default values
    class priority, 161
    class type, 161
    delay distribution, 162
    drop rule, 162
    fork station, 163
    interarrival times, 162
    number of servers, 162
    queue capacity, 162
    queue strategy, 162
    routing strategy, 162
    station type, 162
Demo
    fitting, 223
    workload analysis, 221
Disabled

routing, 68
Distribution
Burst (General), 48, 142
Burst (MAP), 49, 142
Burst (MMPP2), 49, 142
Coxian, 50, 142
Deterministic, 50, 142
Erlang, 50, 142
Exponential, 51, 142
Gamma, 52, 142
Hyperexponential, 52, 142
Lognormal, 53
Normal, 53, 142
Pareto, 54, 142
Phase-Type, 54, 142
Replayer, 55, 142
Uniform, 55, 142
Weibull, 55
DPS scheduling, 39, 72
Drop Rate, 55
Drop rules, 39

Earliness, 56
EDD scheduling, 71
EDF scheduling, 72
Effective Utilization, 56
Equilibrium
with Class Switch, 90
Exponential distribution
CDF, 223
fitting, 223
pdf, 223

Fastest Service routing, 67
FBFU scheduling, 163
FCFS, 149
FCFS (Priority), 149
FCFS scheduling, 39, 69
FCFS-PR scheduling, 72
FCR, 63
see Finite Capacity Region, 63
FCR Total Capacity, 56
FCR Total Memory, 56
Finite Capacity Region, 45, 63
bandwidth contention, 122
capacity, 65
class specific properties, 64
global properties, 64
model example, 119, 122
Firing Throughput, 56
Fitting, 197, 221
exponential, 223
Pareto, 223
Fork Join Number of Customers, 56
Fork Join Response Time, 56
Fork Station, 80
branch probabilities, 82
capacity, 80
Class Switch, 82
distribution of tasks, 82
forking degree, 82
multi-branch Class Switch, 82
random subset, 82
standard strategy, 81
strategies, 81
Forking degree, 82

GPS scheduling, 39, 71
GSPNs, 92

Hybrid models, 93
queueing place, 96

Impatience
Balking, 73
Reneging, 74
Retrial, 101
Installation of JMT, 2
folders used, 2

JABA, 179
all in one graph, 188
convex hull, 184
saturation sector, 183
utilization, 187
JMCH
Results of animation, 174
Markov Chain animation, 175
Modelling Classroom Helper, 169
Routing algorithms, 172
Scheduling algorithms, 170
JMVA, 5
actions, 21
algorithms, 5
command line, 33
import model to JSIM*graph*, 21
input tabs, 6
model definition, 5
model examples, 21
model modification, 18
model solution, 14
Reference station, 11
starting the solver, 5
what-if parameters, 12
XML file, 34
Join Station, 83
Guard strategy, 84
Quorum strategy, 84
Standard strategy, 84
jSIM.log file, 124
JSIM*engine*, 95
event calendar, 96
JSIM*graph*, 39
available distributions, 46
class switch, 88
command line, 124
default values, 99
drop rules, 39
errors and warnings, 103

examples, 115
features, 40
files csv with index values, 111
Finite Capacity Region FCR, 63
graphs of results, 111
initial state, 98
MMPP2 parameters, 49
percentiles of results, 111, 113
reference station, 45
stopping criteria, 60
templates, 136
XML file, 133
JSIM*g*
model creation, 45
JSIM*wiz*, 139
default values, 161
JSQ routing, 67
JWAT
Apache log example, 207
clustering, 210
format definition, 3, 204
input definition, 200, 201, 223
input from file, 202
sampling methods, 202

Lateness, 56
LCFS, 149
LCFS (Priority), 149
LCFS scheduling, 39, 69
LCFS-PR scheduling, 72
Least Utilization routing, 67
LEPT scheduling, 39, 71
Linearizer, 5, 20
LJF scheduling, 39, 70
Load Dependent
functions, 14
routing, 67
station, 8, 227
Load Independent station, 8, 227
Logger
files csv, 2
JMCH, 178
JSIM*graph*, 85

Markov Chain, 169
Max Relative Error, 58, 163
MSER-5 rule, 108
MVA, 5

Non-preemptive
see Scheduling algorithm, 69
Number of Customers, 56, 155

Open model
arrival rate, 45
Optimal population mix, 30

Pareto distribution
CDF, 223
fitting, 223
pdf, 223
Per-class System Power, 56
Performance Indices
Arrival rate, 55
Balking Rate, 55
Drop Rate, 55
Earliness, 56
Effective Utilization, 56
FCR Total Capacity, 56
FCR Total Memory, 56
Firing Throughput, 56
Fork Join Number of Customers, 56
Fork Join Response Time, 56
Lateness, 56
Number of Customers, 56, 155
Per-class System Power, 56
Queue Time, 56, 155
Reneging Rate, 56
Residence Time, 56, 155
Response Time, 56, 155
Response Time per Sink, 56, 155
Retrial Orbit Size, 56
Retrial Rate, 56
Retrial Residence Time, 56
System Drop Rate, 55
System Number of Customers, 56, 155
System Power, 6, 56, 58, 155
System Response Time, 56, 155
System Throughput, 56, 155
Tardiness, 56
Throughput, 56, 155
Throughput per Sink, 56, 155
Utilization, 57, 155
Petri nets, 92
colour, 96
enabling rule, 94
firing rule, 94
immediate transition, 94
inhibitor arc, 94
input arc, 94
marking, 94
output arc, 94
place, 94
timed transition, 94
token, 94
Place Station, 92
drop rule, 93
queue policy, 93
storage capacity, 93
storage strategy, 93
PNML, 97
Arc element, 97
core models, 97
high-level Petri nets, 97
Net element, 97
Place element, 97
Place/Transition nets, 97
symmetric nets, 97
Transition element, 97

Polling
Exhaustive, 75
Gated, 75
K-Limited, 75
Population mix, 123, 183
Power of k choices
routing, 68
Preemptive
see Scheduling algorithm, 69
Processor Sharing, 71, 149
with multiple servers, 71, 149
PS scheduling, 39, 71

QPNs, 92
QQ-Plot, 210, 223
Queue policy, 69
FCFS, 149
FCFS (Priority), 149
LCFS, 149
LCFS (Priority), 149
Processor Sharing, 149
Queue Time, 56, 155
Queueing discipline, 39
Queueing disciplines, 139

R5 heuristic, 108
RAND scheduling, 39, 70
Random numbers generator, 101, 163
seed of, 101, 163
Reference station, 45, 116
in JMVA, 11
Reneging Rate, 56
Residence Time, 56, 90, 155
vs Response Time, 90
Response Time, 56, 90, 155
Response Time per Sink, 56, 155
Retrial Orbit Size, 56
Retrial Rate, 56
Retrial Residence Time, 56
Round Robin routing, 67
Routing, 40, 139
Class Switch, 68
Disabled, 68
Fastest Service, 67
JSQ, 67
Least Utilization, 67
Load Dependent, 66, 67, 70
policies, 67
Power of k choices, 68
probabilities, 67
Random, 67
Round Robin, 67
Section, 67
Shortest Response Time, 67
Station, 84
Routing probability, 63

Scaler Station, 92
Scheduling algorithm
see Queue Policy, 69
BAS, 39, 100
DPS, 39, 72
EDD, 71
EDF, 72
FCFS, 39, 69
FCFS-PR, 72
GPS, 39, 71
Impatience, 73
LCFS, 39, 69
LCFS-PR, 72
LEPT, 39, 71
LJF, 39, 70
Non-preemptive, 69
Non-preemptive (priority), 69
Polling, 74
Preemptive, 69
PS, 39, 71
RAND, 39, 70
SEPT, 39, 71
SJF, 39, 70
SRPT, 72
TBS, 72
Semaphore Station, 91
SEPT scheduling, 39, 71
Service demand, 9
Service Time, 10, 62
Shortest Response Time routing, 67
Simulated time
stop on, 101, 163
Simulation
animation interval, 60, 106, 164
confidence intervals, 58, 106, 108
control, 58
dead index, 109
default parameters, 161
files with index values, 111
graphs, 106
initial state, 60
length, 59
max number of samples, 60
max time definition, 124
maximum duration, 59
parameters, 58
percentiles of results, 111, 113
quartiles of results, 111
results, 105
see also JSIM*graph*, 39
seed, 59
seed definition, 101, 124, 163
spectral analysis, 108
statistics on results, 111
stop time, 60, 101, 163
transient detection, 107
wall time, 59
What-If parallel threads, 102, 164
Sink Station, 68
SJF scheduling, 39, 70
Source Station, 66
Spectral analysis, 108

SRPT scheduling, 72
Station
  class switch, 88, 146
  Deadlines, 76
  delay, 78
  fork, 80
  Heterogeneous Service Times, 77
  Job Parallelism, 77
  join, 83
  load dependent, 8, 75, 78, 227
  load independent, 8, 75, 78, 227
  logger, 85
  M/M/1, 175
  M/M/1/k, 175
  M/M/c, 175
  M/M/c/k, 175
  place, 92
  queueing, 68
  reference, 116
  routing, 84
  scaler, 92
  semaphore, 91
  Server Type, 76
  sink, 68
  source, 66
  Switchover times, 75
  transition, 92
  zero service time, 75, 79
Statistics
  of simulation results, 111
System Drop Rate, 55
System Number of Customers, 56, 155
System Power, 6, 56, 58, 155
System Response Time, 56, 91, 155
System Throughput, 56, 155

Tardiness, 56
TBS scheduling, 72
Templates, 136
  community, 136
  official, 136
  download of, 136
  from JMT official site, 136
  from users' systems, 136
  of a parallel model, 136, 138
  of a three tiers intranet, 137
  of Ceph, 137
  of RAIDs, 137
Throughput, 56, 155
  maximization, 30
  per Sink, 155
Throughput per Sink, 56
Tolerance, 18
Total capacity, 62
Transient detection, 107
  MSER-5 rule, 108
  R5 test, 108
  spectral analysis, 108
Transition Station, 92
  atomic firing, 96
  certain firing, 96
  enabling condition, 93
  enabling degree, 94
  firing outcome, 94
  firing priority, 94
  firing time distribution, 94
  firing weight, 94
  inhibiting conditions, 93
  mode, 96
  number of servers, 93
  timing semantics, 95
  timing strategy, 93

Utilization, 57, 155
  Delay station, 78, 144
  Finite Capacity Region, FCR, 64

Visits, 10
  count, 90
  ratio, 90, 91
  to resources, 90

Wall time
  simulation, 59
What-If Analysis, 60
  example, 119
  parallel threads, 102, 164

XML file
  of JMVA, 34
  of JSIM*graph*, 133

Zero Service Time, 79