The University of Reading

BENCHMARK SYSTEM FOR A STORAGE RESOURCE BROKER

A Dissertation
Submitted In Partial Fulfilment Of
The Requirements For The Degree Of

MASTER OF SCIENCE

In

NETWORK CENTERED COMPUTING,
HIGH PERFORMANCE COMPUTING

in the

FACULTY OF SCIENCE

THE UNIVERSITY OF READING

by

Carsten Koebernick

March 3, 2006

University Supervisor: Prof. Vassil Alexandrov
Placement Supervisor: Dr. Adil Hasan

Abstract

This dissertation deals with the design and development of a measurement system for a "Storage Resource Broker" (SRB). The SRB is a client-server middleware that provides an interface for connecting to heterogeneous data resources over a network. The aim of the project is to enable long-run measurements of the SRB server and standard applications.

The document starts with comparing two state of the art monitoring system to find one that helps with gathering and transmitting measurement data. Afterwards the document focuses on other necessary tools, which help to measure applications and to store them in a relational database. Two basic approaches will be visualised, which will meet the task of the project. The measurement system has been developed to gather measurements from more than one server, so both approaches base on a client server system. Furthermore, measurement graphs have been developed to compare the load and memory efficiency of different machines and measurement cycles. Finally, the measurement system should help the researchers of the SRB to find possible leaks and ease the further development.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The developers of network applications often have to fight more problems with their applications then developers of standard applications. The programme has to be resistant against network interruptions or changes in the connection, such as different latencies. When multiple computers exchange data, it would be good to know, which computer is a bottleneck or has capacity problems. Thus for improving and testing network applications and services, every single computer which is part of the application has to be monitored to find mistakes. In many cases, new programmes have to be developed, which have only the exercise to watch the actual application.

The Council for the Central Laboratory of the Research Councils (CCLRC) has the mentioned problem and needs an application, which is able to monitor every application and one in particular on a Linux operating system. Furthermore, already existent monitoring systems should be considered, if they are able to do the work. The particular application is the Storage Resource Broker (SRB), which has been developed by the San Diego Supercomputer Centre (SDSC). The CCLRC takes part in the further development. The SDSC Storage Resource Broker is a client-server middleware that provides an interface for connecting to heterogeneous data resources over a network. It is able to access different storage systems, such as databases or file servers, and represents the storage outwards as if it woulde be one storage. Further information about the system are described in section 3.1 on page 13.

1.1 Project Description

The project deals with the research, design, and development of a monitoring system to measure the performance and efficiency of the SRB system. The dissertation is created on behalf of the RAL (Rutherford Appleton Laboratory), which is part of the research department CCLRC. The project requires the development of a system to control the running of applications, to monitor, and to collect statistic outputs. This system will be used to benchmark various SRB and Linux applications and will give a brief overview about the capacity and efficiency of the monitored computer. The monitoring system must be able to connect to more than one computer at a time and measure the performance in parallel. Data that will possibly collected are CPU efficiency, the amount of used file descriptors, open disk space and the number of processes. History graphs will be needed to get a better view on the monitoring data. These graphs should be embedded in a Graphical User Interface (GUI).

1.2 Motivation

The CCLRC works on the improvement of the SRB, tries to determine errors, and wants to tune the application. Therefore measuring the performance of a system can be very important to determine bottlenecks. The CCLRC would like to know how the SRB reacts in particular situations. Thus, the motivation behind this project is to find possible mistakes in the SRB and get more feedback from the systems. The developers of the SRB have already written some testing applications, which should bring the SRB up to its limits. For instance, a huge amount of data is copied within the SRB systems and the capacity of the participating computer might be reached. Caused by the structure of the SRB it is not possible to predict on which SRB server in the network will be the most load. So a framework is needed, which starts test applications and is able to monitor more than one computer, which might be influenced by the test applications. Consequently, the target of the project is to help the developers find possible efficiency leaks and help them with the further extension of the Storage Resource Broker by developing an adapted monitoring system.

1.3 Restrictions

An existing monitoring system should be used as far as possible to meet the targets of the project. The data should be stored in a relational database. Furthermore, the database should be small and easy to install. SQLite fits these requirements and will thus be used for data storage. The implementation of further applications has to be done with the scripting language Python. The version of Python is restricted to 2.2.3 to ensure compatibility to older working environments. The GUI needed to draw the diagrams has to be implemented with Tkinter, an Application Programming Interface (API) between TK scripting language and Python.

All other applications that are needed for the project must be installable without superuser rights in Linux. This means every user has to be able to install Python, Tkinter, the existing monitoring system, and SQLite on every Linux machine used for measurement. For a better extensibility and reusability of the programmed code, an object-oriented approach has to be used for the implementation.

1.4 Document Structure

This dissertation consists of nine chapters. The first and current chapter gives a short introduction into the topic of the project. The second chapter “Monitoring Systems” presents two state of the art monitoring systems and explains them. At the end, both will be compared and the one used in the project will be exposed. The third chapter explains all additional technologies used to support the existing monitoring system and to create the measurement system. Therefore, the programming languages, database and the SRB server will be explained in detail.

The “Analysis” chapter describes the exercise of the measurement system and shows two different approaches to meet the tasks of the project. Afterwards the design shows the chosen architecture, the modules, object models and the database models of the measurement system. The “Implementation” chapter describes the Installation of the needed tools and describes five code examples in detail. So, the “Design” chapter explains, what the measurement system does and the “Implementation” section how it has been implemented. The last main chapter “Tests” shows two different cycles of the measurement system and describes the results. The “Conclusion” chapter summarises the whole work

again, shows what has been achieved, not fulfilled and over-fulfilled during the project. The "Future Prospect" helps with the further development of the measurement system by suggesting a few improvements. At the end of the document is the Appendix, which consists of a short documentation of the programmes and the implemented code.

The experienced reader does not need to read the "Technologies" chapter, because it explains database and programming bases, which he might already knew. But the following chapter assume that the reader know about the bases

Chapter 2

Monitoring Systems

In a time where administrators have to fight with networks getting bigger and bigger by connecting to other networks or by embedding them into a Grid[1], monitoring systems become more and more important.

A monitoring system constantly monitors different sections of a computer. Conceivable applications are network, performance and service monitoring. Network monitoring is used to constantly supervise computer networks. Mostly the in and output will be measured. Performance monitoring can be used to watch the efficiency of a server. For instance if an administrator wants to know if a server operates on its full capacity or is the server able to run another service. Service monitoring controls running applications and the system is able to handle eventually defects.

A monitoring system for Linux uses different sections of the Linux file system. There are plenty of possibilities to get performance and statistic information from a Linux operating systems. The most common way is using the "proc"-directory. The "proc"-directory has a dynamic content, that means whenever someone wants to get information from the proc-directory, the Linux-Kernel creates the information. So, the information of this directory depend on the actual state of the running machine. By creating the information dynamically the directory needs no memory capacity on the hard disk. The monitoring tools often use this folder to get their information. Every process, which is running in Linux has a process-ID. During the process is alive, a directory with the specific process-ID will be stored in the proc-directory. A lot of information can be found about the process like, how much cpu and memory usage needs the process. Sometimes monitoring systems use

[1] Grids are resources of many separate computers connected in a network (often the internet) to solve large-scale computation problems

standard supervision programs like "vmstat","lsof" or "ps". "ps" can be used to get information about running processes, "vmstat" returns information about the virtual memory of the system and "lsof" lists open files. But at the end these supervision programmes use as well the proc file system to get their information. The advantage is that the programmes are nearly independent from the used Unix derivate [1]. The proc file system has different structures on different Unix derivates and sometimes it is not even existent.

A monitoring system has to be easy to use and needs to be extensible. The current monitoring systems are often used for big networks, clusters or Grids. The information of the monitored computers are mostly stored in a database or sometimes ordinary files are used. Furthermore monitoring systems provide a possibility to produce diagrams for instance to watch the CPU efficiency of the last month. If errors have occurred on the machine, the graphs might give a reference to the problem.

Two systems are currently state of the art. Ganglia [2] and Nagios[3] are widely used to monitor computers in a network, cluster or Grid.

2.1 Ganglia

Ganglia is a scalable distributed monitoring system for high performance computing systems such as Clusters and Grids, but it can be used just for one computer or small networks as well. It was developed by Matthew L. Massie and is an Open Source development project under General Public License (GPL) [4]. Ganglia uses several common

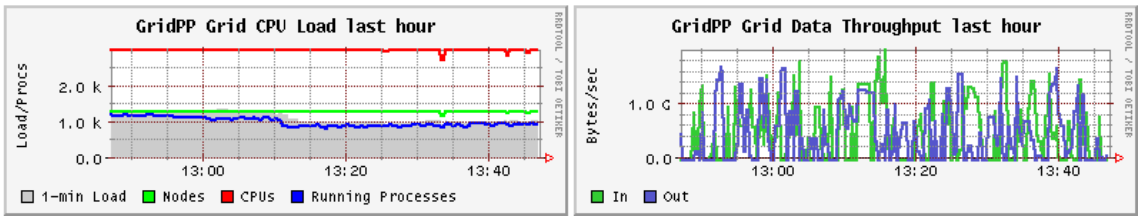


fig. 2.1: Ganglia Output of University Karlsruhe, Germany

techniques like eXtensible Markup Language (XML) for data presentation. It is a widely used application, which runs on over 500 clusters around the world, for instance it is used by the Wikimedia Foundation to monitor their servers [5].

Ganglia consists of two daemons, a PHP web interface and some little helper tools. A Ganglia monitor needs to run on each node in the network. The monitor gathers the values for various metrics such as CPU load, free memory, disk usage, network I/O (Input/Output), operating system version, etc. These metrics are sent through the network and will be used by the front end node to generate historical graphs. In addition to metric parameters, a heartbeat message from each node is collected by the ganglia monitors. When a number of heartbeats from any node are missing, the web page will declare it as "dead".

The three main components of the Ganglia Toolkit are in detail:

Ganglia Monitoring Daemon (gmond) The standard monitoring daemon in Ganglia is the gmond. It is a multi-threaded daemon, which runs on every node that should be monitored. Multi-threaded applications run more more than process or thread [2] in parallel. The daemon monitors several variables, which have been set within a config file or with the gmetric application. The gmetric application allows the user to set its own metrics by passing a new metric name and its value. Standard variables which can be set in the gmond.conf for a Linux distribution are:

Metric name	Description
boottime	System boot timestamp
bytes_in	Number of bytes in per second
bytes_out	Number of bytes out per second
cpu_aidle	Percent of time since boot idle CPU
cpu_idle	Percent CPU idle
cpu_nice	Percent CPU nice
cpu_num	Number of CPUs
cpu_speed	Speed in MHz of CPU
cpu_system	Percent CPU system
cpu_user	Percent CPU user
disk_free	Total free disk space
disk_total	Total available disk space
load_fifteen	Fifteen minute load average

Continued on next page

[2] A thread is a so called lightweight process. Many threads share resources and the used memory area with each other.

Metric name	Description
load_five	Five minute load average
load_one	One minute load average
mem_buffers	Amount of buffered memory
mem_cached	Amount of cached memory
mem_free	Amount of available memory
mem_shared	Amount of shared memory
mem_total	Amount of available memory
mtu	Network maximum transmission unit
os_name	Operating system name
os_release	Operating system release (version)
part_max_used	Maximum percent used for all partitions
pkts_in	Packets in per second
pkts_out	Packets out per second
proc_run	Total number of running processes
proc_total	Total number of processes
swap_free	Amount of available swap memory
swap_total	Total amount of swap memory
sys_clock	Current time on host

Table 2.1: gmond variables

The same table is available by using the gmond command with the parameter "-m". Furthermore it is possible to pass the gmetric tool a Bourne Again Shell (bash) script. This bash script has to return a value, which will be passed to the gmond daemon. This is a feature of the Linux command line and not of gmetric, but it is the best way to pass calculating functions to gmond. More information about bash can be found in the technology chapter in section 3.5 on page 20. The values, which have been set by the user, need to be updated by himself. That means whenever a user wants get current monitoring data, he has to execute the gmetric command and pass a new value to gmond. But it is also possible to set up a cron job with the crontab command. The crontab command, found in Unix and Unix-like operating systems, is used to schedule commands to be executed, frequently. So the measurement data of Ganglia could be updated with a cron job up to every minute. Here is

an example for a gmetric command, which passes a return value of a bash script to the gmond daemon:

```
gmetric --name time --value 'date | awk '{ print \$4 }'
--type string
```

This command gets the time from the commandline tool `date` and put it into the gmond metrics.

The gmond daemon transmits the information via User Datagram Protocol[3] (UDP) messages in eXternal Data Representation[4] (XDR) format or sending XML over a Transmission Control Protocol[5] (TCP) connection. The simplest possibility to get the data from gmond is to a use a telnet query, which returns the monitoring data in XML. If someone has a gmond daemon on his computer system, the following command would return the monitoring data:

```
telnet 127.0.0.1 8649
```

The command queries the local network device, and returns the information if there are any messages on Port 8649. If Ganglia is running the monitoring will be structured in XML format.

Ganglia Meta Daemon (gmetad) Federation in Ganglia is achieved by using a tree of point-to-point connections over particular cluster nodes to gather the state of multiple clusters. At each node in the tree, a Ganglia Meta Daemon (gmetad) polls frequently a collection of child data sources. The collected XML data will be parsed and saved to a Round-Robin database. The gathered XML data will be spread out over a TCP socket to the other clients.[2]. For more information about the Round-Robin database, please have a look at the homepage of "RRDtool" [6], which is used in the Ganglia toolkit.

Data sources, which should be monitored, may be either gmond daemons, particular clusters, or other gmetad daemons, representing sets of clusters. Multicast[6] channels spread the data over the network to share the monitoring data with other nodes.

[3]UDP - minimal connectionless network protocol
[4]XDR - communication standard between server and clients for data exchange
[5]TCP - reliable connection-oriented network protocol
[6]Multicast - one node sends data to multiple nodes

Ganglia uses the multicast channel to get the information from a group of nodes in the network. A particular IP address space (224.0.0.0 bis 239.255.255.255) has to be used for multicast connections.

Ganglia PHP Web Frontend The Ganglia web frontend provides a view of the gathered information via real-time dynamic web pages. Most importantly, it displays Ganglia data in a meaningful way for system administrators and computer users. It is possible to group the information by clusters and networks, to get an overview about parts of the network.

Every tool can be configured over a configuration file: gmond.conf, gmetad.conf and conf.php. The structure of the monitoring system allows using only parts of the toolkit, because not every administrator needs gmetad to get information about a whole cluster. Mostly the gmond daemon is enough to supervise small networks. Furthermore not every one likes the PHP webpage and wants to draw its own diagrams with a standard GUI.

2.2 Nagios

Nagios is server based monitoring system, which is widely used and very popular. Companies such as Dell and Cisco use it for their own systems. Nagios consists of a server, which provides information for a client. So on every machine, which needs to be monitored runs a server. It is possible to create network hierarchies, which allows the system to stop the monitoring of resources, which cannot be reached anymore. That means if a network node, for instance a router, has been disconnected, all measurement queries for that node and the nodes behind it, will be stopped at once.

Nagios combines many monitoring features of single programmes in one big package. It is a network and system monitoring application. It watches hosts and services, which were specified by an administrator. It is able to send out alerts, if something went wrong or if particular events have happened. Nagios was originally designed to run on Linux, but it should work with the most Unix derivates as well. The resource, which shall be monitored, is called service check in Nagios. These service checks are connected over a plugin manager to Nagios. Nagios has the following additional features [3]:

Monitoring of network protocols Protocols like POP3 (Post Office Protocol Version 3) or SMTP (Simple Mail Transfer Protocol) can be monitored. An overview of sent

mails can be created to count and compare the amount of transmitted and received mails. But a lot more network protocols are conceivable such as ICMP (Internet Control Message Protocol) or NNTP (Network News Transfer Protocol).

Simple plugin design The design of the Application Programming Interface (API) for Nagios allows users to easily develop their own service checks. The service check can be written in any installed programming language. It just have to be executable and return 0 for "OK", 1 for "Warning", 2 for "Critical" or 3 for "Unknown". The most plugins are written in Perl, Bash and C, but it is absolutely not important and a developer can use the programming languages he likes most. Furthermore a lot of service checks are available in the Internet. For example the “NagiosExchange Portal”[7] provides many plugins for free.

Network host hierarchy As already mentioned, Nagios is able to define a network host hierarchy using "parent" hosts, for example a Router. This allows the detection of broken hosts and the differentiation between hosts that are down and those that are unreachable.

Contact notifications The four different states “OK”, “Warning”, “Critical”, and “Unknown” can start particular events. An event handler can be written by the user to react on particular states. It is possible to send mails to the administrator or to use any user-defined function. For instance if a web server has been monitored and the state is "Critical" the server could be restarted automatically and the administrator just gets the information that something has happened.

Automatic log file rotation A log file can be archived if it reaches a particular size or age. This feature allows an automatic archiving of the logging content.

Support for implementing redundant monitoring hosts A monitoring host of Nagios can be installed redundant on two hosts. If one of them fails, the other takes on with the monitoring task. This enables an automatic maintainance of Nagios. For instance, if the primary host, which runs Nagios, fails or when portions of the network became unreachable, the system will still work on another machine.

Optional web interface A web interface can be optionally configured for viewing the current network status, the last notifications, problem history. Besides the overview, the log file can be edited within the web interface.

Normally the measurement data will be stored in two log files, but older versions of Nagios are available with database support for PostgreSQL and MySQL. There are binary packages available for the Linux distribution Debian, Suse and Red Hat. The newest version Nagios 2.0 abandons on databases, but with the PerfParse Toolkit[8] it is again possible to store the data in MySQL. Nagios needs a lot of configuration getting it to work. A sample configuration file, which is called "minimal" is 300 rows long and configures Nagios for the use of only eight service checks.

2.3 Comparison

Nagios has one particular disadvantage, which is to restrictive for the development of the measurement system. Only a Linux root user, the so called superuser, which is able to do everything under the Linux operating system, is allowed to install Nagios. That means whenever a Nagios system needs to be installed on a system, a user with full rights is needed to install the monitoring system. It is possible to use Nagios without User permissions, but for the installation are Superuser rights necessary. It might be possible getting it installed without root rights, but a lot of investigation and time would be needed, because no documentation could be found, where someone has done it. Furthermore Nagios is a lot more complicated than Ganglia and difficult to install. Ganglia has the advantage, that it is divided in three main parts, gmond, which is mostly used for standard networks, gmetad for clusters and the PHP web page to draw diagrams in a browser.

Hence, it follows that the project will use Ganglia to get the monitoring information from the nodes. As mentioned in this chapter, Ganglia uses a Round-Robin database. The project restrictions say that a relational database should be used for data storage. Therefore only gmond will be used to gather monitoring data and further implementations are needed to store the data in an SQLite database.

Chapter 3

Technologies

After the decision had been made to use Ganglia as the system to gather the monitoring data, further research and analysis was needed. This chapter will begin with the explanation of the functionalities of the Storage Resource Broker. It is important to know how the SRB works and which information are needed to monitor the SRB. Furthermore, the chapter explains the details of the prescribed applications and the associated tools. The scripting language Python is also prescribed for the project and some peculiarities of the language will be shown. Afterwards a brief overview about Bourne Again Shell will be given, which helps to monitor the SRB and standard applications. Besides that, the measurement data must be stored in a relational database. So, the fundamentals of database systems will be explained within a section about the Structured Query Language (SQL) and the used database SQLite. The last step will be to emphasise on the abilities of the XML, which is used by Ganglia to provide the monitoring data.

3.1 Storage Resource Broker

The management and access to data, which are stored in different storage systems by different manufacturers is very often a problem of computer centres and IT offices. The users are not interested in the way the data are stored, they just want to have one or two point of references to store and load their data. The Storage Resource Broker allows storing the data not only with typical attributes of a file system. The files can be stored with attributes such as the user information, version number and file specific information, for example parameter of the file. The SRB creates a global name space, which allows splitting the

user from the storage infrastructure. User and applications can organise their data company wide, without being involved saving it on physical data storage. The administrators can store the data where and as they like it, without influencing applications and users. The SRB can access nearly every possible storage system such as relational databases, tape libraries or file servers. Furthermore, it is able to access every attribute of a file through the Meta data CATalogue (MCAT). The MCAT stores the attributes like physical storage location, file attributes, access rights and all user defined attributes of every file. A user can connect to the network with SRB client applications. An SRB client can

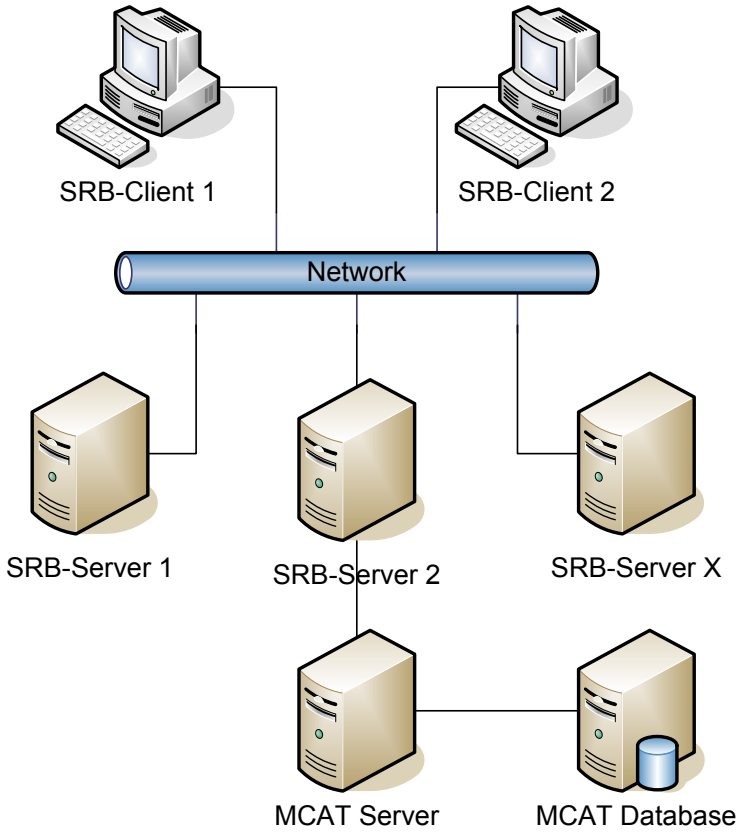


fig. 3.1: SRB network

be a set of UNIX commands called Scommands[1], which have been developed to get the same comfort as in normal UNIX environment. For instance an SRB command "Sgrep" has been migrated from the UNIX "grep" command and has similar functionality. Other access possibilities are GUIs such as the web client MySRB [9], or Windows Client inQ

[1] http://www.sdsc.edu/srb/scommands/index.html

[10]. The picture 3.1 on the last page shows the connection between the SRB-clients, the SRB-server, the MCAT Server and the MCAT database. The whole system can run on a single computer, which is fine for testing purposes. The MCAT database is the metadata repository that provides a mechanism for storing information used by the SRB system. It stores the internal data needed by the SRB system and the user metadata regarding data sets are stored by the SRB.

The SRB has a lot more features, which should only be mentioned:

- Creating and administrating data replications for fast recovery, load sharing and caching
- Transparent migration of data, without changing the view of users and applications
- Security and Backup of data if someone made unintended modification or errors
- Security mechanisms, for example encrypted data transfers and secure authentication methods like Public Key Infrastructure (PKI).
- Scalability, which allows the handling of millions transactions per day
- Mechanisms to minimize latency during the transfer
- Management of access and logging methods
- APIs allow the integration and adjustment of the SRB in any work environment

3.1.1 Scommands

Using the Scommands needs a connection to a remote or local SRB server. To establish the connection, two files are needed on the local machine, which connects the host to the SRB server. The files ".MdasAuth" and ".MdasEnv" are hidden in the ".srb" folder, which is located in the user home directory of Linux. The ".MdasAuth" file contains the password, which enables connecting to the SRB. The ".MdasEnv" file consists of the following information:

```
mdasCollectionName '/homezone/home/srbadmin.homedomain'
mdasDomainName 'homedomain'
```

```
srbUser 'srbadmin'
srbHost '127.0.0.1'
srbPort '5544'
defaultResource 'srbdisk'
AUTH_SCHEME 'ENCRYPT1'
```

The most interesting information are, that the server is running on its own machine ("127.0.0.1") and is connected over network port 5544, which is also called SRB port. Furthermore, it contains of the user name, domain and the collection. An SRB collection is similar to a folder or directory in a file system. The collection is an object that contains other collections or data objects. It organises data objects into a logical hierarchy. Therefore, the hierarchy can be accessed easily. Every SRB user must be a member of one domain. Domains group individuals together that are located on one physical site or office location. The Authentication scheme depends on the used system, which could be a Grid, than particular Authentication Methods will be used, for instance the Grid Security Infrastructure (GSI) [11]. The "defaultResource" points to the first SRB server, which should be used.

So before using the Scommands both files need to be in place. The "Sinit" command can be used to start the connection to the SRB and with "Sexit" the connection will be closed. A further list and an explanation of the Scommands can be found on the SDSC website [12].

3.2 User Management of Unix based systems

The project needs all necessary applications installed with user rights. Therefore, it is important to give a short explanation of the way UNIX deals with its users. The Multi-user system UNIX allows setting privileges for different users. For instance, a usual user is not allowed to stop the machine or delete the complete hard disk. Furthermore, every user has ownership rights, which allow him to prevent his files from other users. The output of the bash command "ls -l" will help to explain the ownership of files:

```
drwx------ 2 carsten staff  2048 Jan  2 1997  private
drwxrws--- 2 root   admin  2048 Jan  2 1997  admin
```

```
-rw-r----- 2 carsten staff 12040 Aug 20 1996  myfile
drwxr-xr-x 3 carsten user   2048 May 13 09:27 public
```

Field 1 The first field is a set of ten permission flags. The first flag distinguish directories "d" from normal files "-". A "-" says that no flag is set. The next 3 flags are the permission of the owner. "r" for read, "w" for write and "x" for execute. The same permission flags come than for the Group and for Others. The position where x would normally go is sometimes a "s" flag, which is called set-UID or set-groupID flag.On an executable program with set-UID or set-groupID, that program runs with the effective permissions of its owner or group. The set-groupID flag means for a directory that all files, which are created inside that directory, will inherit the group flag. When directory has no flag, all files take the primary group of the user that created the file. This issue is important to people that try to keep a directory as group accessible. The subdirectories of the main directory also inherit the set-groupID flag.[13]

Field 2 - link count describes how many times the file is linked to other positions

Field 3 - owner of a file (user) is set in field 3

Field 4 - the associated group of the file

Field 5 - size of the file in bytes

Field 6-8 - date of last modification. The format varies between UNIX derivates, but consists always of three fields.

Field 9 - name of the file

When a new user was created, he gets his own folder in the home directory, which has the name of the user. The user is allowed to install his programmes and to store his files there. Hence, all applications, which are needed for the project, have to be installed somewhere in the home directory.

3.3 Python

> "Python is an interpreted, interactive and object-oriented programming language."[2]

It is completely written in C and has been developed in the beginning of the Nineties by Guido van Rossum at the "Centrum voor Wiskunde en Informatica" in Amsterdam. Originally, it has been developed for the distributed operating system "Amoeba" and consists of many features from the programming languages Lisp, Smalltalk, FP and Unix-Shells. The name of the programming language is based on the English comedy group "Monty Python", so connections in Python documentations are volitional.

Python has been advanced and is one of the most important scripting languages and becomes more and more popular. The programming language is easy to learn and the syntax is manageable. Python consists of big standard library, which means the programmer needs less effort to achieve much. It is comparable with Perl, PHP or Java. Classes, Modules, exceptions and dynamic data types can be used.

There are interfaces to system calls and libraries, as well as to various windowing systems such as X11, Motif, Tk, Mac, MFC, wxWidgets, and Qt. New built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface. The Python implementation is portable: it runs on many brands of UNIX, Linux, on Windows, OS/2, Mac, Amiga, and other platforms. If a particular system has no Python support, yet it may be supported later, as long as there is a C-Compiler for this system. Sometimes nobody tried to compile Python for a particular system before. Python can be used free for commercial and non-commercial use, but it is copyrighted.

Python has peculiar syntax, which nearly needs no brackets. It uses in instead of brackets indention to switch between looping constructs and normal commands.

```
<statement1>
while <test>:
    <statement2>
```

This forces the user to write clean and human readable code. Semicolons are not necessary and are proscribed. Python is completely object-oriented and a lot of modules and classes

[2] http://www.python.org/doc/Summary.html

are provided in the Internet. So whenever one heard somewhere of "Python, batteries included" it means the programmer should search in the library before starting to reinvent the wheel.

For a better understanding of the attached Python scripts in B on page XXI, it is necessary to explain the meaning of the underscores before attributes and operation names. Python classes cannot be restricted to public, protected or private. However, the underscore helps to avoid name collisions between attributes of different classes. A double underscore is a pseudo private prefix. The name of the operation, for example "__gui()", will be changed during the interpretation to "_<class-name>__gui". This avoids any name collisions. A single underscore prevents the attributes to be sent between files. So if a new file imports classes from another file in this way "from "filename" import *" the attributes are not available in the new file. Every other attribute name is visible for everyone. The constructor of every class is the "__init__" function, where every member variable should be declared. The constructor will be started, when an object has been created. The attribute "self" is needed in every member function of a class and allows the access to all other member functions and variables. For instance the constructor can be called with "self.__init__()".

3.4 Tkinter

Tkinter is a portable library for the construction of graphical user interfaces and is part of the standard library modules of Python. It is implemented as a thin-object oriented layer on the basis of Toolkit (Tk). Tk is an extension of the Tool Command Language (Tcl). Tk is an open source, cross-platform widget[3] toolkit, which is a library of basic elements for building a GUI. So all the possibilities which provides Tk can be used with Tkinter in Python. Tkinter is one of the most commonly used GUI toolkits, because it is portable between Mac, Windows and Unix. Furthermore it is easy to learn like Python, because it consists of only a few widgets, but it is mostly enough for smaller applications. Unlike Java every scrollbar must be connected to a particular widget by the programmer, himself. So a lot more code is needed to get small things to work. Tkinter is not only used, because it is portable, it is also pretty fast, which is sometimes a big disadvantage of interpreted languages.[14]

[3] components of a GUI

3.5 Bash

UNIX allows a different command line interpreter, which are called shells. It is an important connection between the User and operating system. UNIX shells execute commands sequentially, therefore they are often used as scripting languages to write little helper applications to relieve the everyday life. The Bourne Again SHell (Bash) is part of the GNU.

fig. 3.2: GNU-project logo

GNU is a recursive acronym for "GNU's Not UNIX", a project to write a free Unix like operating system. Bash is fully Bourne Shell (sh) - compatible, which is parent of the most new shells. The bash extends the shell with features of the Korn Shell (ksh) and the C Shell (csh). The bash offers the functionality to use it for programming and interactively. Because of the compatibility with Bourne Shell, all older scripts will run without modifications under the bash. The improvements offered by the bash are, for example command line editing, an unlimited size can be used for the command history or indexed arrays of unlimited size. Further implements the bash, a Job Control system, which allows the user having the system work on a job in the background, while he is working with the keyboard.

3.6 Database system

A database system consists of two parts, the database, which is an organized collection of data, and a database management system (DBMS), which is used to query and manage the database. The database system has to store a huge amount of data and must be able to react on requests from applications and users. Databases store collection of records in a systematic way, so particular information is easier to find and can be ordered and grouped by using requests. There are different types of databases, which differ in their data model.

3.6.1 Database models

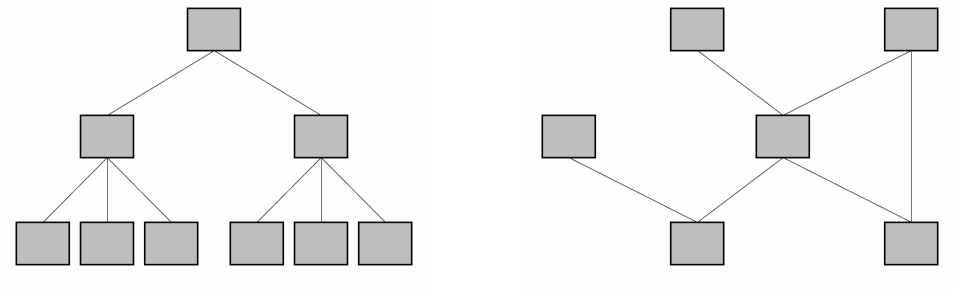
3.6.1.1 Hierarchical model

The hierarchical model is the oldest data model that is not often used in databases anymore. The databases, which are based on the hierarchical database model, use a tree model for displaying data.

The object types have definite connections to each other. The data model has a root, which is connected to his subordinates. The structure is efficient for the computer, but is not possible to do any changes without varying the organisation of the data. Requests need to orientate on the tree structure.

3.6.1.2 Network model

DBS, which are based on the network model, create a structure by using networks. It consists of no root objects and so every object type can have more than one connection to other objects. Connections have to be labelled.



(a) Hierarchical Database Model (b) Network Database Model

fig. 3.3: Database models

3.6.1.3 Relational model

The relational model is one of the most important and widespread data models. The main part of this model is a table (relation). The data is managed in two-dimensional tables, which are connected by primary keys and foreign keys. Primary keys are unique and

define every row in a table. The foreign keys are primary keys from other tables, which build the connection between different a tables. The relational model allows the using of easy describing database languages such as the Structured Query language (SQL). The section 3.6.3 on the following page will explain more details of SQL.

3.6.1.4 Object-oriented model

The content of an object database are objects. An object is a summary of connected attributes in a record. The objects are similar to the objects of the object-oriented programming. The advantage of the object database model is that the model can be used to interleave the objects and to build a breakdown structure. For instance: Car $\longrightarrow$ Motor $\longrightarrow$ V-belt

3.6.2 Entity Relationship Model (ERM)

The entity relationship model is used to describe a part of the real world in the data modelling. The ERM is the preliminary stage for the design of relational database schemes. It consists of a graphic or a description for the elements of a database. Therefore, the ERM consists of two main terms, entities and relationships. The entity is an item, which represents things of the reality, for instance a person, a book, or a house. The relationships represent the connection or dependence between two entities. For example, the person "lives in" the house or the book "is on" the shelf. The last term, which is often used in case of ER-models, is the cardinality. This is the possible number of connected entities to one relationship. So there could me more than one book on the shelf, but the same book cannot be on more than one shelf. One differentiates between one-to-one, one-to-many and many-to-many relationships. The many-to-many relationship needs a further development, because it cannot be solved in a database schema. Therefore, join tables will be used to solve it. That is an additional table, which consists only of foreign keys and one primary key. Nevertheless, one should avoid using them and the developer should try to find another solution. The Entity Relationship model is mostly used in the beginning, when the database gets a concept. The planner gets an overview about the data, which will be needed in the database to consider every possibility and avoid redundancies. Furthermore there are different notations for drawing ERM models. The most popular ones are:

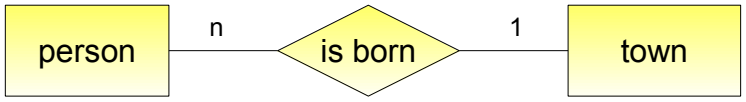


fig. 3.4: Chen notation

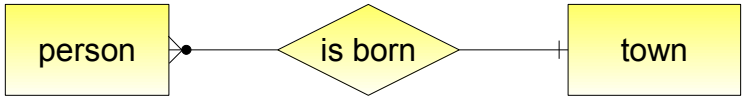


fig. 3.5: Bachmann notation

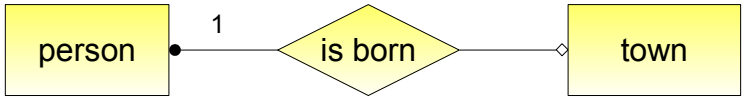


fig. 3.6: IDEF1X notation

Other notations are Must-Can Notation (MCN), numerical notation or (min,max) notation.

3.6.3 Structured Query Language - SQL

SQL is a declarative computer language for relational databases. It is the main standard language for relational databases and allows independence from applications. SQL has an easy syntax, which is based on the English colloquial language. Unfortunately, every manufacturer has its own extension for SQL and so the programmer has to make necessary adjustments. Many popular DBS like MySQL, MSSQL (Microsoft SQL Server), PostgreSQL, DB2 or SQLite use parts of the SQL implementations.
SQL is divided into four partial languages:

3.6.3.1 Data Definition Language (DDL)

The DDL defines the database scheme. Three commands are part of the language: "CREATE" can be used to generate, "DROP" to delete and "ALTER" to change a table. Furthermore, primary and foreign keys can be set during the creation and the changing. As mentioned before, not every database implementation allows every command. For instance the SQLite database, which will be further explained in subsection 3.6.4 on page 26, is not able to handle foreign keys and the "ALTER" command is not implemented in SQLite version 2.

Examples:

```
CREATE TABLE house( h_id INTEGER PRIMARY KEY,
colour varchar(50), address varchar(50) NOT NULL)
```

The query creates a table house, which has a primary key h_id, a colour and address with a string length 50. The address must have a value, because the column is not allowed to be empty.

```
ALTER TABLE house DROP address
ALTER TABLE house ADD zip_code varchar(10) NOT NULL
```

The “ALTER” queries erase the column address and add a new column zip_code (some databases for instance Oracle 9 version 2, allow the same with "RENAME", than just the name would have been changed by using the address column as the new zip_code column without loosing the data).

```
DROP TABLE house
```

The “DROP” query deletes the table house.

3.6.3.2 Data Query Language - DQL

DQL is used to request data from the database. It consists only of one command: “SELECT”. The command has many abilities by joining tables and creates new output tables. Particular attributes can help to get a smaller set or sorted part of the database. The attributes are [15]:

- "WHERE" identifies particular rows and retrieves them. ("SELECT * FROM house WHERE colour = "yellow" " $\longrightarrow$ returns every yellow house)

- "GROUP BY" combines rows with related values into elements of a smaller set of rows
- "HAVING" is used to identify, which of the "combined rows" are to be retrieved. (used in combination with “GROUP BY”)
- "ORDER BY" sorts the output data by a particular column (can be sorted ascending ("ASC") or descending ("DESC")

Furthermore there are little functions called aggregate functions to do calculations during the retrieval of data ("SUM,MIN,MAX,COUNT") For example, you could also use the "SUM"-function to return the name of a department and the total sales of the associated department. The "HAVING" clause will filter the results that only departments with sales greater than 1000 Pounds will be returned:

```
SELECT department, SUM(sales) as "Total sales"
FROM order_details
GROUP BY department
HAVING SUM(sales) > 1000
```

3.6.3.3 Data Manipulation Language - DML

The DML consists of three commands: "INSERT", "UPDATE", "DELETE". The function "INSERT" is used to insert values into a table. "UPDATE" changes values and "DELETE" erases values of a table. These functions are used to change the content of the table.

Examples:

```
INSERT INTO table_name (name1, name2)
values (value1, value2)
```

This inserts value1 into column name1 and value2 into column name2.

```
UPDATE table_name SET name1=value2
```

It changes every column name1 to value2.

```
DELETE FROM table_name
```

This query deletes every row from a table.

3.6.3.4 Data Control Language - DCL

The DCL sets user rights for a database. The function "GRANT" gives the user particular rights and the function "REVOKE" takes the rights away.

Examples:

```
GRANT SELECT,UPDATE ON table_name
```

This “GRANT” query allows the user to change and get data from the table.

```
REVOKE SELECT ON table_name FROM PUBLIC
```

This “REVOKE” query denies every user, who is not stored in the system with other rights, to see data.

The database, which will be used during the project, has not implemented any DML queries, because the only rights management of an SQLite database is the Unix right system. Therefore, the possibilities and restrictions of SQLite need to be explained in the following section.

3.6.4 SQLite

SQLite is a programme library, which contains a relational database system. It supports a lot of standard database functions like transactions, subselects, views, trigger and user-defined functions. The database was originally developed for embedded computing and so it can be used with every important programming language. However, it is also provides a standard interface, which is usable over the command line and shell scripts. SQLite is different from most other SQL database engines. It was primarily designed for simple applications. Therefore, SQLite is simple to administer, operate, embed in larger programmes, and easy to maintain and customize. Many people like SQLite because it is small fast and reliable. Reliability results from the simplicity of SQLite. With less complication, there is less to go wrong. SQLite is has its strength and weaknesses, depending on the things the user wants to achieve. It is not made for high concurrency and has no fine-grained access control or a huge amount of inbuilt functions. Nevertheless, in many situations it is the right choice, where for example someone needs a data storage, which has to be portable and fast accessible. SQLite cannot compete with Oracle or PostgreSQL and do not want to do it. It just rules in another area.[16]

3.6.5 Pysqlite

Pysqlite is an API for SQLite. Pysqlite makes SQLite available to Python programmers and allows using the advantages of the small database. It stays compatible with the Python DataBase Application Programming Interface (DB-API) specification 2.0 as much as possible. The DB-API 2.0 has been defined in order that the developers try to programme similar Python modules for accessing databases. Therefore, whatever database a developer chooses, he can use the same Python commands to access the database. SQLite and Pysqlite are good alternatives to other databases like MySQL or Postgres, because it is fast, small and easy to use. [17, 18]

3.7 XML

The extensible markup language is a standard for the creation of machine and human readable documents in tree structure. The World Wide Web Consortium (W3C) has defined this standard. XML defines the rules for the body of such documents and applications have to follow these rules and specify their own details.

XML has been arisen out of the Standard Generalized Markup Language (SGML) project. SGML has a lot more options than XML, but it is too complicated for most applications, so XML became the main standard for markup languages. The names of the structure elements of an XML-application can be chosen freely. An XML element consists of or describes different data, for example texts, graphics, or abstract data types. The main reason of using XML is to differ between data and the presentation of the data. For instance, graphs can be shown in tables or in pictures, but both can be stored in XML format. An XML file consists of elements, attributes, text, instructions and commentaries:

elements - consists normally of a start-tag and an end-tag, which surround the content (e.g text), So called Empty-Element-Tags have just one tag and surround nothing.

```
<TAGNAME>contents</TAGNAME>
<TAGNAME\>
```

attributes - are mostly used for optional information in the starting and empty tags.

```
<TAGNAME attribute-name="attribute">
```

instructions - are used to identify the XML file .

```
<?target-name parameters ?>
```

commentaries - are optional and can be used to describe the code.

```
<!-- commentary-->
```

3.7.1 Structure and Rules

The structure of an XML document is nearly unrestricted. There are just some elements, which always have to be in the file. There is a beginning tag, which identifies the file as an XML document.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

The version attribute identifies the version of the XML file. The encoding attribute indicates which character set is used. For instance UTF-8 announces a Unicode[4] character set, where every character has a particular number. UTF-16 is a bigger character set, which additionally is able to store Chinese and Japanese symbols.

The second predefinition, which has to be in an XML file, is the root tag. This tag consists of all other following tags in the document and only one root tag is allowed.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
   <other tag>content</other tag>
</root>
```

The whole content of the document can now be stored between the root tags. Tags are case sensitive and so if using different cases for the Start and End Tag, the XML Parser would return an error. Furthermore, it is only allowed to begin a tag with an underscore and a letter. The following characters can be letters, numbers, points or lines. The last main rule is that no nested tags are allowed:

```
<Tag1>content1 <Tag2>content2 </Tag1><Tag2>
```

This would be the correct form:

```
<Tag1>content1 <Tag2>content2 <Tag2></Tag1>
```

[4]Unicode website - www.unicode.org

3.7.2 Parser

If an application wants to use the information of an XML file, the user needs to parse the file to get the structure and the information. Therefore, the application must have an XML-parser. The programmer can write his own parser, but that is mostly not necessary, because most programming languages got the two main parser already implemented. The two different parsers are Simple API for XML (SAX) and Document Object Model (DOM).

3.7.2.1 SAX

The SAX parser reads the XML file, sequentially. That means every tag will be loaded after another and the tree structure will be available after the complete file has been parsed. The information is returning one after another during the parsing. This parsing algorithm needs less memory, because only current tags will be loaded in the memory. Hence, it is useful for large files with a lot of content.

3.7.2.2 DOM

The DOM parser is the second way, to analyse XML files. It loads the complete file in the memory and at first analyses the tree structure. The advantage is that, all information is available in a hierarchical tree structure and can be accessed at the same time. The big memory requirements could be a problem, because the requirements are proportional to the file size. The problem is not the amount of memory which is been used during the parsing. The latency, between loading the file and the first access can be very long and disturb the application, which is using the XML-Parser.

3.7.3 Meta languages

Another difference between parsers is validating and non-validating parsers. These parsers try to check before reading the document, if the XML syntax is well formed. This means if the original intended XML form looks like the XML form in the actual document. Therefore, Meta languages exist, which describe the XML-format before the content follows. The most popular Meta languages are the Document Type Definition (DTD) and the XML-schema (XSD).

3.7.3.1 DTD

The Document Type Definition defines the building of an XML file. Further it defines the structure with a list of legal elements. The DTD can be declared inside of the XML document or as an external reference to another file.

Internal DOCTYPE declaration If the DTD should be included in the XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

The document type starts with the root element, which is followed by the element declarations in brackets.
Example:

```
<?xml version="1.0"?>
<!DOCTYPE point [
  <!ELEMENT point (x_value,y_value)>
  <!ELEMENT x_value      (#PCDATA)>
  <!ELEMENT y_value      (#PCDATA)>
]>
<point>
  <x_value>10</x_value>
  <y_value>20</y_value>
</point>
```

The DTD above is interpreted in the following way. !DOCTYPE point defines that this is a document of the type point. !ELEMENT “point” defines that the point element has two elements: "x_value,y_value". !ELEMENT x_value and !ELEMENT y_value defines both to be of type "#PCDATA"[5].

External DOCTYPE declaration The external doctype declarations contain of a reference to another file:

```
<!DOCTYPE root-element SYSTEM "filename">
```

3.7.3.2 XSD

XML-Shema is the modern possibility to describe the structure of XML documents. XML-Shema allows to restrict the content, elements and attributes. The restrictions can

[5]#PCDATA - parsed character data, which means PCDATA is text that will be parsed again, because it may consist of elemenents and attributes

be done with regular expressions[6]. XSD is more complex than DTD, but therefore a lot more complicated especially the specification. XSD is using an XML format, which allows using XML tools to process them.

Example:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="point">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="x_value" type="xs:int32"/>
        <xs:element name="y_value" type="xs:int32"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

This is the XSD description for the same XML example as in section DTD. As mentioned before, it is a lot more complex and need a bigger header. The "complexType" shows that the tag "point" has more elements between its start and end Tag. The other tags x_value and y_value are so called "Simple tags", because they do not contain other elements.[19]

There are many other meta languages such as Document Structure Description (DSD) or REgular LAnguage for XML Next Generation (RELAX NG), which are adapted to new requirements of the developers. However, the DTD is for most tasks complex enough and is still used. For instance, the monitoring system Ganglia uses DTD to describe the XML output of the gmond daemons.

[6]Regular expressions describe subsets of strings

Chapter 4

Analysis

The following chapter will provide a further insight into the project. The target of the first section is to analyse Ganglia and to point out what is needed to embed it into the project. Mainly the interfaces are interesting for the further development. The second section deals with the SRB server and what is necessary to measure the SRB server application. Then two possible approaches to solve the exercise will be given. Both have the ability to connect to several servers and get the monitoring information from these computers. Weaknesses and strengths of the solutions will be presented and a decision will be made.

4.1 Ganglia

The measurement system implemented during the project uses parts of the existing monitoring system Ganglia. It needs to collect and transfer the measurement data between the machines in one network. For this task, the gmond daemon can be used and configured to use standard Host-to-Host connections to provide the measurement data of the machines. The gmond daemon will run on every machine, which takes part of a measurement. The “gmond.conf” of the gmond daemon will need an extra parameter to transmit the information to one machine, which measures the other ones.

```
udp_send_channel {
    host = <hostname>
    port = 8649
  }
```

The change of the hostname allows to send the measurement data to any host in the network. As mentioned in the technology chapter, gmond has some standard metrics, which can be set in the configuration file "gmond.conf". But the project needs only a few values like the CPU system efficiency or the processes running on the machine. But some other values, which are not preconfigured, are needed by the project. It should be possible to measure applications and to get values like open filedescriptors, the CPU efficiency and the used memory of the application. The "gmetric" tool is able to create and fill new metrics into gmond. Therefore it is used to pass the application measurements to gmond. It is necessary to set up the name of the new value, the type and the value itself. The type can be string, int8, uint8, int16, uint16, int32, uint32, float and double. The value can be passed directly by giving a string or an integer. But it is also possible to pass the return value of Bash script to gmetric. The following lines will give an example for gmetric command line:

```
~/bin/gmetric -c ~/sbin/gmond_test.conf --name srb_fd
 --value `./num_fd.sh 5544` --type int16
```

Unfortunately gmetric has to be executed frequently to refresh the data in gmond. This means one has to get the monitoring data from the system and put them with gmetric into Ganglia, frequently.

The measurement system will get the measurements by using a telnet query. The query returns an XML string which sorts the measurement data by its hosts:

```
<GANGLIA_XML VERSION="3.0.1" SOURCE="gmond">
<HOST NAME="localhost" IP="127.0.0.1" REPORTED="1139739824" TN="6" TMAX="20" DMAX="0" LOCATION="unspecified"
      GMOND_STARTED="0">
<METRIC NAME="cpu_speed" VAL="598" TYPE="uint32" UNITS="MHz" TN="42" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/
      >
<METRIC NAME="mem_total" VAL="1295744" TYPE="uint32" UNITS="KB" TN="42" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="
      gmond"/>
<METRIC NAME="proc_total" VAL="111" TYPE="uint32" UNITS="" TN="2" TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="machine_type" VAL="x86" TYPE="string" UNITS="" TN="42" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/
      >
<METRIC NAME="cpu_user" VAL="13.0" TYPE="float" UNITS="%" TN="2" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_nice" VAL="0.0" TYPE="float" UNITS="%" TN="2" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_system" VAL="3.8" TYPE="float" UNITS="%" TN="2" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_idle" VAL="81.9" TYPE="float" UNITS="%" TN="2" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="load_one" VAL="0.47" TYPE="float" UNITS="" TN="42" TMAX="70" DMAX="0" SLOPE="both" SOURCE="gmond"/>
</HOST>
<HOST NAME="192.168.10.100" IP="192.168.10.100" REPORTED="1139739826" TN="3" TMAX="20" DMAX="0" LOCATION="unspecified"
       GMOND_STARTED="0">
<METRIC NAME="cpu_speed" VAL="598" TYPE="uint32" UNITS="MHz" TN="3" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="mem_total" VAL="1295744" TYPE="uint32" UNITS="KB" TN="3" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond
      "/>
<METRIC NAME="proc_total" VAL="111" TYPE="uint32" UNITS="" TN="3" TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="machine_type" VAL="x86" TYPE="string" UNITS="" TN="3" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="cpu_user" VAL="13.5" TYPE="float" UNITS="%" TN="3" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_nice" VAL="0.0" TYPE="float" UNITS="%" TN="3" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
```

```
<METRIC NAME="cpu_system" VAL="3.9" TYPE="float" UNITS="%" TN="3" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_idle" VAL="77.9" TYPE="float" UNITS="%" TN="3" TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="load_one" VAL="0.53" TYPE="float" UNITS="" TN="3" TMAX="70" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="bash_cmd" VAL="bash" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"/>
<METRIC NAME="gkrellm_cmd" VAL="gkrellm" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="
    gmetric"/>
<METRIC NAME="bash_cpu" VAL="0.1" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"/>
<METRIC NAME="gkrellm_cpu" VAL="1.1" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"
    />
<METRIC NAME="bash_fd" VAL="12" TYPE="int16" UNITS="" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"/>
<METRIC NAME="gkrellm_fd" VAL="13" TYPE="int16" UNITS="" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"/>
<METRIC NAME="num_of_apps" VAL="2" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"/>
<METRIC NAME="bash_mem" VAL="0.4" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"/>
<METRIC NAME="gkrellm_mem" VAL="1.0" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric"
    />
<METRIC NAME="hostname" VAL="linux.site" TYPE="string" UNITS="" TN="0" TMAX="60" DMAX="0" SLOPE="both" SOURCE="gmetric
    "/>
</HOST>
</GANGLIA_XML>
```

The XML string has no starting and ending tag for the values. Only attributes have been used for the measurements.

Finally the Ganglia daemon has to be embedded by connecting to its interfaces. The measurement system needs to configure, start and stop gmond. The gmetric tool is needed to pass custom values to gmond and afterwards the measurement data must be queried with telnet. Therefore it will be a main task to find the best interaction between Ganglia and the measurement system.

4.2 SRB System

The measurement system has no direct access to the SRB system. It is not the task of the measurement to start, control and stop the SRB system. But the test applications should create load on the SRB server, which can be monitored with the measurement system. The test applications will start a particular number of Scommands to increase the load. A problem, which has not been touched yet, is that more than one SRB-Server could run on one machine. But the SRB servers on one machine belong to different SRB systems. Each SRB system uses a unique network port. That means every SRB server running in one system uses the same port. Figure 4.1 on the following page illustrates, two different SRB systems on one machine.
Therefore the measurement system needs to know the port of the SRB server that should be monitored. The ".MdasEnv" file in the home directory needs to be parsed and the SRB port can be extracted. Afterwards the system is able to measure the correct SRB server. The SRB server application starts several processes, for example SRB master and SRB

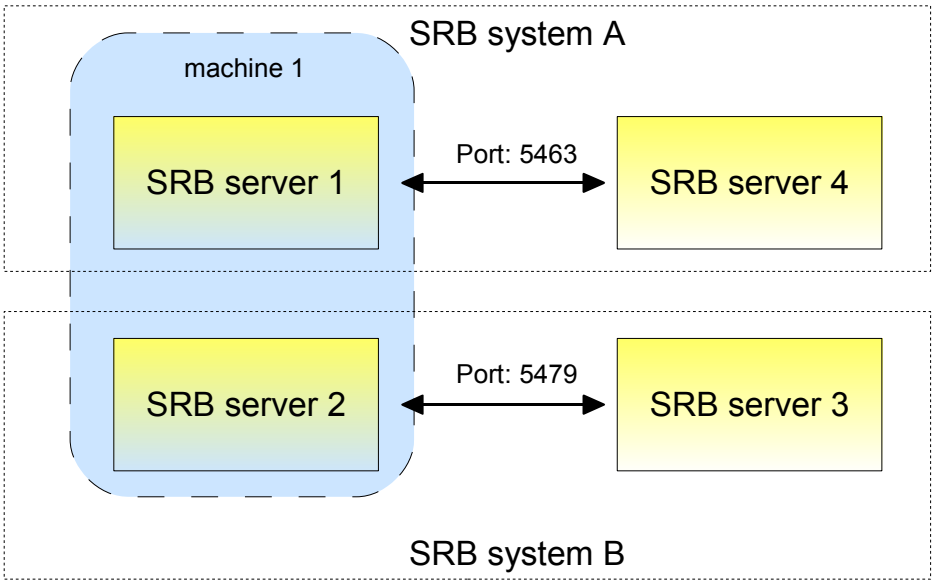


fig. 4.1: SRB server connection

server. Therefore the values of all processes need to be summed up. It can be done with the help of Bash scripts and data will be transmitted to Ganglia with gmetric.

4.3 Basic Approach

The Ganglia section showed that a gmond daemon has to run on every machine and it has to know where to send the measurement data. Furthermore a script is needed, which measures the applications on the machines and submits the data to gmond with gmetric. This could be done with Linux cron jobs, which are able to run scripts in a particular time interval. But still one has to activate and deactivate cron jobs, because the measurement system is supposed to be stopped after the measurements have been finished. Furthermore cron schedules are not precise enough. It is possible to start tasks every minute, but not for example every 15 seconds. The cron daemon is a highly sophisticated application, but does not fit to the measurement system.

But still someone has to refresh the performance data. Therefore, a script has to run on every machine which measures the performance data of the applications and inserts them into gmond with gmetric. But the user of the measurement system might not want to change the hostname on every machine and set up the applications, which should be

measured. It would be better to have one point of reference to configure the measurement system. Therefore a client server connection is useful to provide the scripts with the application names. The monitoring client would be the machine, which starts and

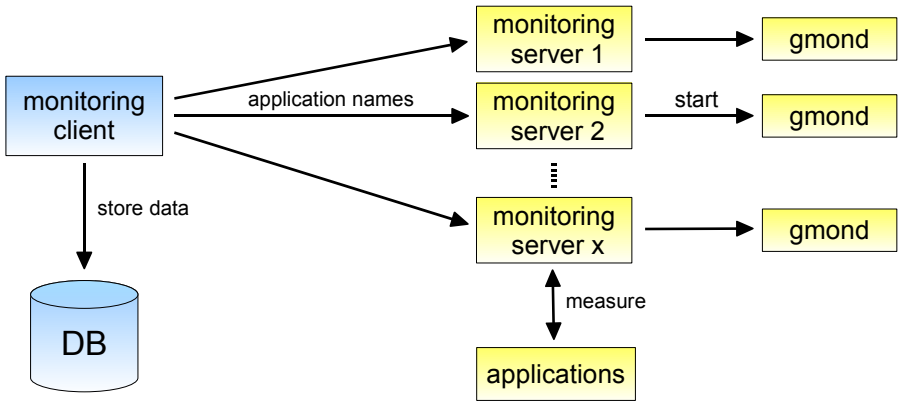


fig. 4.2: Client-Server connection

stops measurement cycles and is the single point of configuration. The client sends the configuration data to the server. The client will get the measurement data and store them in an SQLite database. The server will get the application names and the SRB port to distinguish between different SRB systems. So the client must be able to connect to more than monitoring server to transmit configuration and to receive measurement data. Picture 4.2 gives an overview of the fundamentals of the measurement system approach: The figure shows that the server script starts and stops gmond, measures the applications and passes the values with gmetric to gmond. But the question is, how gets the client script the measurement data. The client can either get the gmond data from its local gmond daemon, which is able to catch the data from the other gmond daemons in the network or the monitoring client gets the data directly from the servers by using the already open connection which was used to send configuration data. Both possibilities are conceivable and need a more precise analysis in the following section.

4.4 Solutions

4.4.1 Solution with Network Functionality of Ganglia

The first possible solution is that the client gets the data directly from gmond. That means every gmond daemon, which runs on a server machine, sends the data with the daemon to the client gmond. By changing the IP address and the port in the send channel of the gmond.conf, the measurement data will be sent directly to the client gmond. The

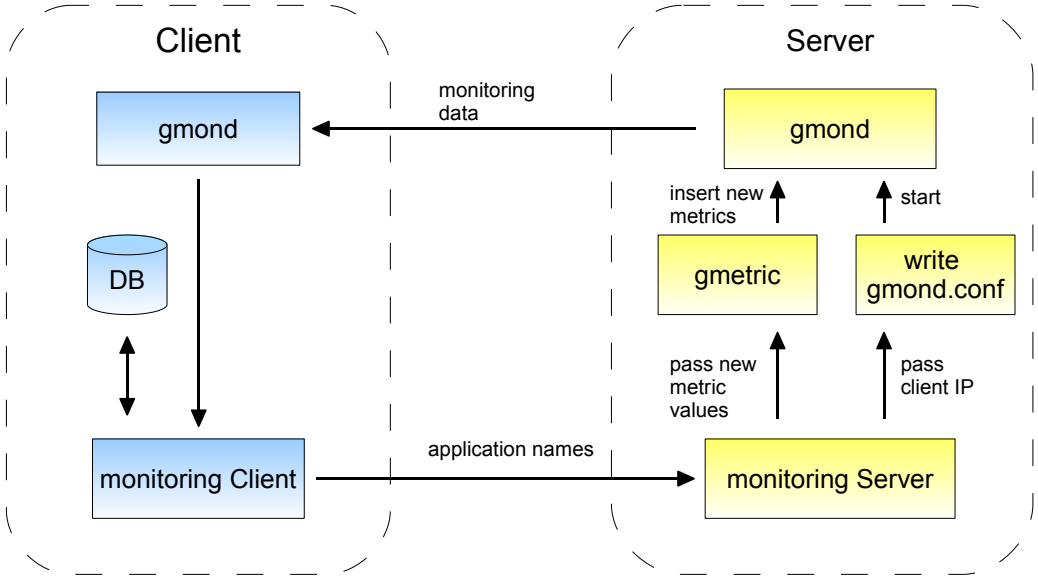


fig. 4.3: Solution 1

process starts with the client, which sends the application names and if needed the SRB port to the server machines. Then the server takes the IP address from the client and changes the Send channel in the gmond.conf. Afterwards the server will start gmond with the changed gmond.conf. The server can now collect the measurement data of the application with particular scripts and use gmetric to submit them to the gmond daemon. So whenever a monitoring server updates measurements in gmond, the server gmond will transmit the current data to the client gmond. The monitoring client can now access the data by querying its local gmond. The query returns an XML string with the monitoring data separated by the host name of the machines. The last step is to prepare the monitoring data and store it in the SQLite database.

4.4.2 Solution with Data Transmission over the Socket

Another solution is to use less of the Ganglia system. One could abandon on the network possibilities of gmond and use the own connection, which is needed to send the application names. Ganglia would still be used as the main resource for the monitoring data, but every monitoring server fetches the measurements from gmond by itself. Therefore a send channel in the gmond.conf is set to localhost. This means every gmond daemon keeps the measurements stored on the local machine. After the monitoring server has collected the XML output from its gmond, the data will be forwarded to the client as an XML string. When the client has received the measurement data from every server, the data will be parsed and stored in the database as well.

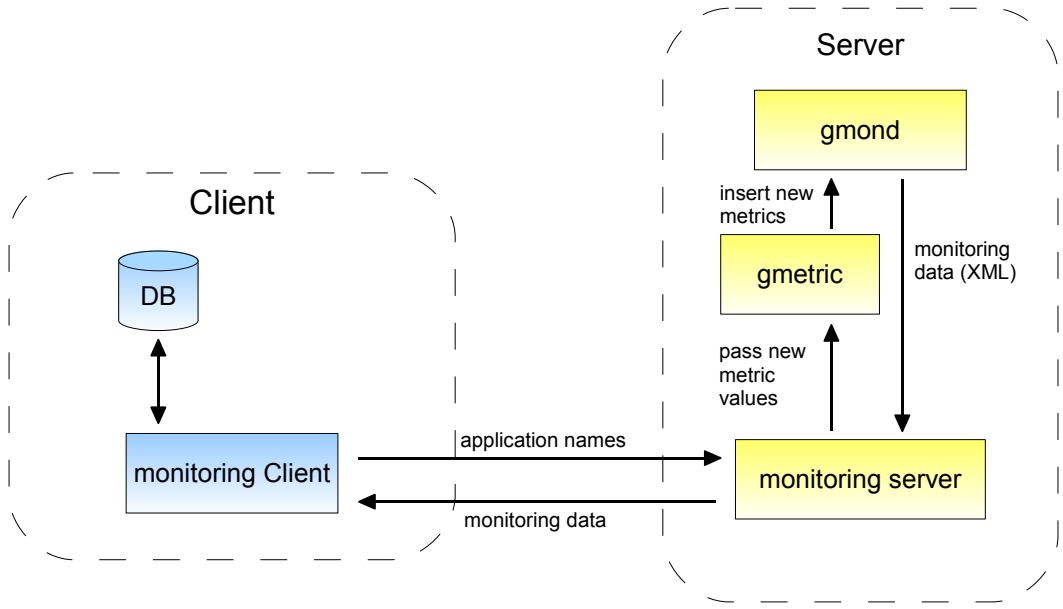


fig. 4.4: Solution 2

4.4.3 Comparison and Decision

Solution number one might seem more complicated at first sight, two network connections are involved: one between the monitoring server and client and one between the gmond daemons. However the implementation is far easier, because the main network transmission is outsourced to Ganglia. The second solution would need a full duplex

socket. This means the socket needs the ability to send into both directions. Furthermore the gmond daemon sends XML strings, which can become bigger in the further development of the projec. Thus a lot of data might have to be sent through the socket connection. Furthermore the monitoring client would receive an XML string from each monitoring server. As a result, each single XML stream has to be parsed, before the data can be stored in the database. It can become a real bottleneck, when many servers have to be monitored. The receiving, parsing and storing of the data can be parallelised with threads, but the SQLite database could make problems with threaded connections. Regarding to the SQLite-FAQ[20], a single database connection should not be used in more than one thread.

> "It is never safe to use the same sqlite3 structure pointer in two or more threads."

The problem has been acknowledged during some tests with SQLite. It occurred with one database pointer and four threads, which use the same pointer. It was possible to reproduce the message "library routine called out of sequence", which leads to the conclusion that the SQLite database has not enough security mechanisms to handle these connections.

So the first solution would work fine with a single thread to handle the parsing and storing of the measurements. It is possible, because the monitoring data of all servers are in one XML string. Furthermore the usage of Ganglia would not make any sense in the second solution. The measurement data for the applications have been given to gmond and the server would query the same data from gmond. This is a circle, which makes no sense, because the server could abandon on Ganglia and create its own XML string. Consequently the first solution had been used for the design and development of the project.

Chapter 5

Design

The design chapter starts with database models and the explanation of the relations and attributes. The database is just a part of the client application, but its explanation is necessary to understand the following programme sequence section. This section will provide an overview about the tasks during a measurement. Afterwards the module concept, that is behind the design of the applications will be shown and described. The next step is to go further in the development and show the object models for the client and server. The methods will show a better insight to the applications.

5.1 System Architecture

The analysis has shown that a client-server approach is the best design for the measurement system. Therefore two applications are needed to meet the tasks of the project. Both applications are affected by the external application monitoring system Ganglia and need to embed it as good as possible.

5.1.1 Monitoring Server

Task The monitoring server script is responsible for the operation of the gmond daemon on the machines, which should be measured. The server gets the application names and the SRB port from the client to measure particular applications on its machine. For the transmission of the measurement data, the server has to start the gmond daemon and configure the send channel of the "gmond.conf". Therefore the server writes the IP address

to the host parameter in the "gmond.conf". To provide the measurement values of the application, the server script has to run gmetric commands. Three particular application measurements have to be provided: the percentage of CPU load caused by the application, the amount of used file descriptors and the percentage of memory, which has been used by the application. Caused by the complexity of the SRB, the measurements are different from the measurements of standard applications. As mentioned before different processes will be started by the SRB server. The processes with the command name SRB master and SRB server will be summed up to one value. Therefore, the server has to distinguish between the measurement of SRB servers and standard applications. Furthermore, the monitoring server has to pass the measurements of the applications in a particular time interval, which is more frequent than the client tries to get the data from gmond. This means as long as the client is connected, the server updates the data repeatedly. When the monitoring client requests the server to stop measuring, the server stops the gmond daemon and the connection to the client.

Interfaces Figure 5.1 shows four different interfaces of the server script. The gmond daemon will be started with a standard execution command and the server needs only the process id of gmond, to kill the application after the measurement has finished. Gmond needs a parameter, "-c", which passes the path to the gmond.conf, which configures gmond. The measurements of the applications will be done by Bash scripts, which re-

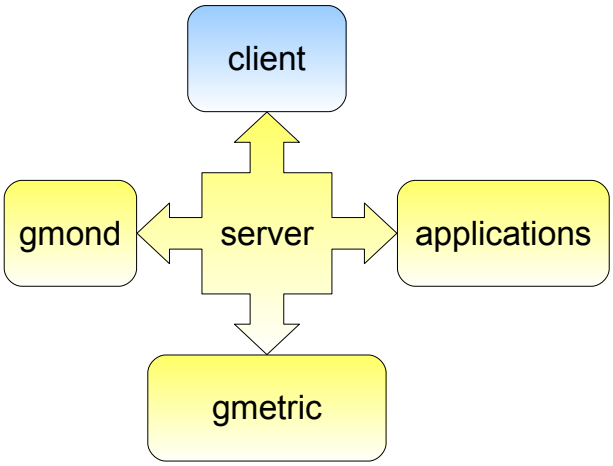


fig. 5.1: Server interfaces

turn the measurement value. The Bash scripts use standard commands such as "ps" and

"ls". The following "gmetric" command line is an example for passing a return value of an SRB Bash script to gmond:

```
~/bin/gmetric -c ~/sbin/gmond_test.conf --name srb_fd
 --value `./num_fd.sh 5544` --type int16
```

The last interface is a socket connection from the client, which is used only into one direction. The server opens a TCP socket and the client connects to it. TCP sockets acknowledge every sent package and assure that the data will be transmitted without errors. The server closes the connection to the client after the client sent it a message to do so. Afterwards the server will listen for a new connection.

Design The server application is a command line tool, because it is able to run without interaction between the user and the programme. It is a small application, which needs less load and configuration. The server has to open a port, so the client application is able to connect to it. The server accepts one parameter "-p", which opens the port for a client connection. The server will open port 5000 if no parameter is given.

5.1.2 Monitoring Client

Task The client is more complicated than the server is. Before the client connects to the server, it needs more configurations. This will be done by a configuration file, which can be changed by the user. Furthermore, the client has to execute a test application, which creates load on the SRB system, before it starts a new measurement. The test application has to be started and stopped by the client application. When the user starts a measurement, the client has to connect to the servers and send the applications names and the SRB port. Afterwards the client starts its own gmond daemon and queries the measurement data with telnet. The output from telnet is an XML string, which will be parsed by the client. Now the measurement has to be stored in the database.

Besides the performing of measurements, the client provides the user the ability to browse through old measurements and draw these in diagrams. The diagrams can be saved as a Postscript file, which can be used for printing.

Interfaces The client has to handle four interfaces as well as the server. The figure 5.2 on the next page provides an overview on the interfaces. The first interface, which will be used by the client, is the configuration file, which provides the settings for a measurement. The configuration file will be parsed directly after the client has been started. It has the following structure:

Table 5.1: Configuration file content

Configuration Tag	Description
project	
name	project title of the project table (reusable)
description	project description of the project table
test	
application	applications which must be measured
name	name for the test table (reusable)
description	description for the test table
tester	
surname	surname of the tester
forename	forename of the tester
email	Email address of the tester
server	
host_x	server host or ip address
port_x	server port
measurement	
srb	SRB flag to measure the SRB server or not (boolean)
application	application which should be started during the test
poll_time	time interval between two measurements
quantity	maximum number of measurements

The test, project and tester tags are tables of the database and will be explained in the next section of this chapter. The server tag sets the host and port of the server, where the client should connect. The configuration file can have up to nine servers with port and host. The "x" at the end of "host_x" and "port_x" has to be substituted by a number between one and

nine. The application parameter of the test tag sets the application names, which should be measured on the server side. A semicolon can be used to separate the application names, if more than one application should be measured. The measurement tag consists of the SRB flag, the test application and the poll_time. The SRB flag can be "1" for the measurement of an SRB server and 0 for a standard measurement without an SRB server. The test application parameter sets the application, which should be started before a measurement. The poll_time parameter sets the time interval between two measurements. That means, when client has done a telnet query to get a new measurement, it waits for the "poll_time" interval before querying again. The second interface is the connection to the server, which

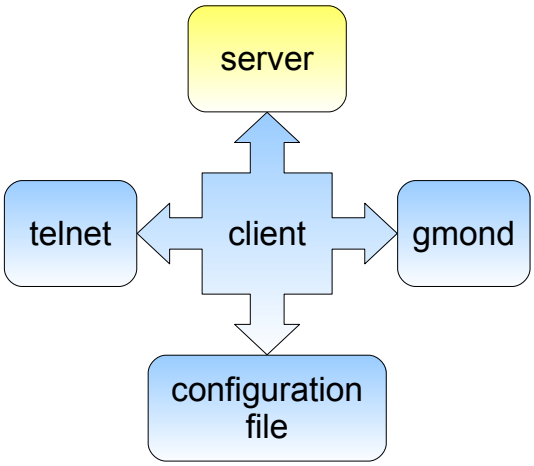


fig. 5.2: Client interfaces

is logically a TCP socket. The client has to send the server the tasks. This means if the client sends the application names, the server knows it has to start gmond and measures the applications. Furthermore the client stops the connection by sending a "clientquit".

Before querying telnet for measurements, the client has to start its gmond daemon to receive new measurements. The telnet query can be executed by using the localhost device of the client machine and the TCP port of gmond. This port is by default 8649:

```
telnet localhost 8649
```

The command returns the XML string with the measurement data of the hosts.

Design First of all the client application has been divided into two separate applications. One is console based and allows the user to measure other machines. The other

application provides a Graphical User Interface (GUI) interacting with the measurement system. Both allow the measuring of the machines and the storing of measurement data in a database. However, the console application provides no graphical functions like the history diagrams. It just allows a small view on the database tables. Apart from the interface to the client application, both can use the same functions to do measurements, parsing, storing in the database, to connect to the server application and to query the gmond for measurement values.

The console client has mainly the task to do measurements, without any interactions. During the measurement, the console client will draw console tables to acknowledge the arrival of a new measurement. For further information about the measurements, the user should use the GUI. After the measurement has been stopped manually or automatically by reaching the maximum number of measurements, the application closes. A small feature of the console client will be the console tables. It is a little bit cumbersomely to go through the tables by using ids, but sometimes no GUI might be available. Therefore, it is just for exceptional circumstances.

The GUI has two different main dialogs with Single Document Interfaces (SDI). SDI handles only one main window. An example for an SDI GUI is the Microsoft Internet Explorer 6.0. The opposite is the Multiple Document Interface (MDI), which is often used by word processors like the Mozilla browser 1.5. This allows handling more than one Document at the same time.

The first dialog is to configure the measurement and browse through the tables. Therefore, it consists of two different frames. This dialog will start the second dialog if the user wants to measure and draw the dynamic graph or to see an old measurement cycle in a static graph. Furthermore, the GUI can start and stop measurement cycles. The measurement cycle will be supported with a dynamic graph. The diagram changes, whenever new values have been written to the database. The user can configure the configuration file over the GUI. This is a feature, which abandons the user from an extra editor to change the configuration of a measurement cycle.

5.1.3 Measurement Sequence

The programme flow chart 5.3 on the following page should give another view on the interaction between the server and the client application. In addition to the tasks, which

have been already mentioned in the last two sections, the flowchart diagram will show the distribution of some tasks. Because the server and the client need an additional thread during the measurement sequence. The first important thing is to start the monitoring server

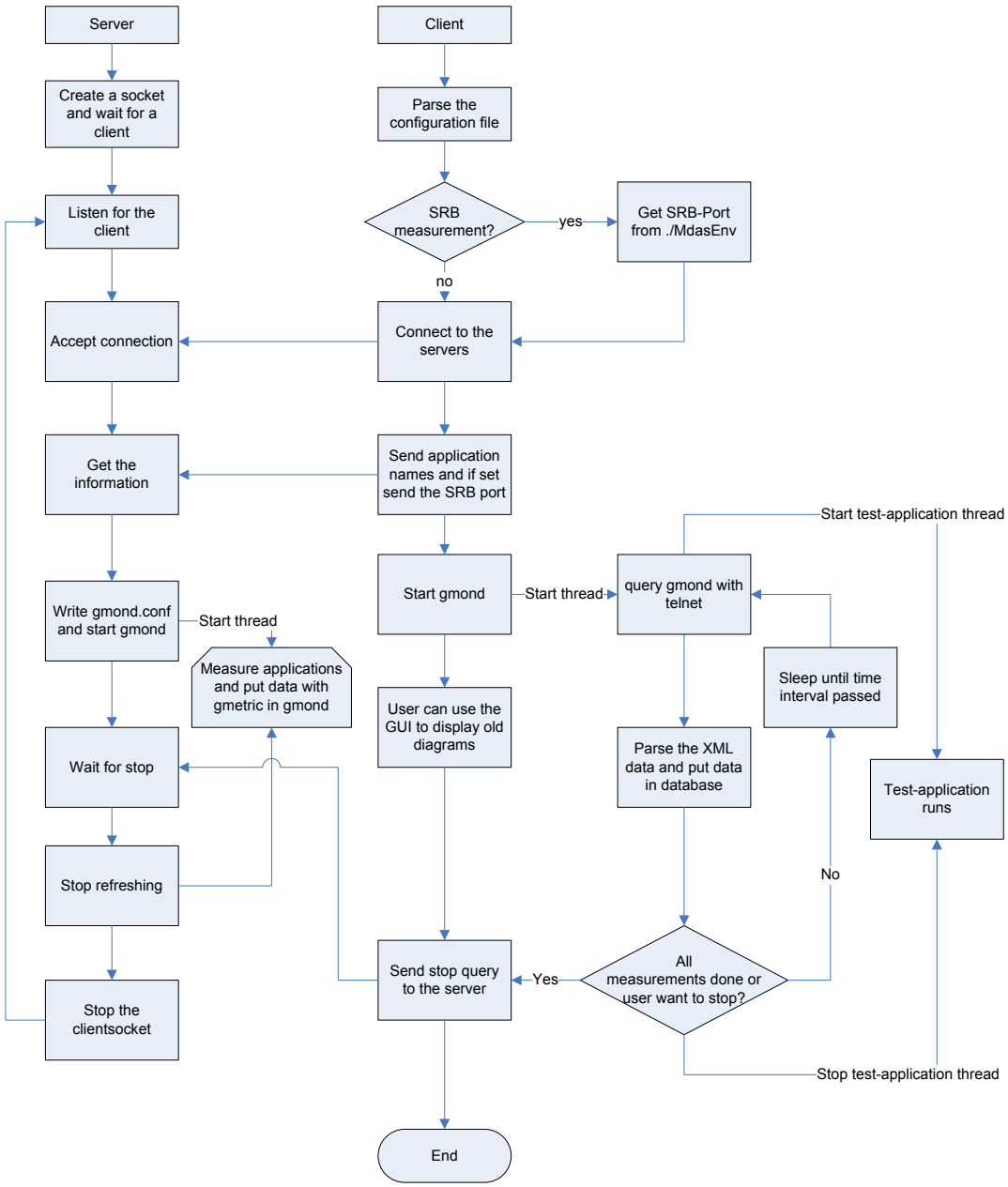


fig. 5.3: Programme flow chart

on the server machine, so the client is able to connect to it. Afterwards the server waits for a connection from the client. Before the client is able to start any connections, one has

to know to which servers the client should connect. Therefore, the client uses the configuration file and parses it. Afterwards, the client is configured for a new measurement cycle.

The next step is to test if the servers are reachable. If not the measurement will not start. If the SRB flag is set the client programme has to get the SRB port from the ".MdasEnv" file from the ".srb"-folder in the home directory. Now the client can connect to the servers and send the application names and the SRB port. The server will write the "gmond.conf" with the IP from the client and start gmond. Furthermore, the server measures the applications, which have been given by the client and passes them with gmetric in gmond. The client can now get measurement data from the gmond daemon running on the client side.

The client queries gmond via telnet for new values from the servers. Furthermore, the client gets standard metrics, without any application metrics from its own machine as well. All values will be parsed and afterwards saved in the database. When the first measurements have arrived and all servers sent data, the test application will be started.

The measurements of the SRB servers are now available for the history diagrams. Whenever a new measurement arrives the history diagram will be updated and show all measurements of the measurement cycle. This will be continued until the maximum number of measurements has been reached or the user at the client side stopped the measurement cycle. During the measurement, the server updates the gmond, frequently. Nevertheless, the server has to wait for a "clientquit" message from the client. Therefore, the monitoring of the applications has been outsourced to a new thread. So the server can do both, wait for a message from the client and update the data for gmond. The client starts also a thread for the measurement cycle. This allows the user to work with the GUI at the same time.

When the client has finished the measurement cycle, it sends the "clientquit" message and the server stops at first the thread and then the connection to the client. Afterwards the server waits again for a new connection from a client.

5.2 Database

One of most intensive research topics was to find an appropriate way of storing the measurements in the database. The Entity Relationship Model (ERM) 5.4 on the next page

will give the first prospect of the database. All connections between the entities are one to many relationships.

5.2.1 Entity Relationship Model - ERM

The Entity Relationship model in figure 5.4 gives only a short insight to the finally implemented database. The measurements need to be stored in a particular hierarchy, so the user

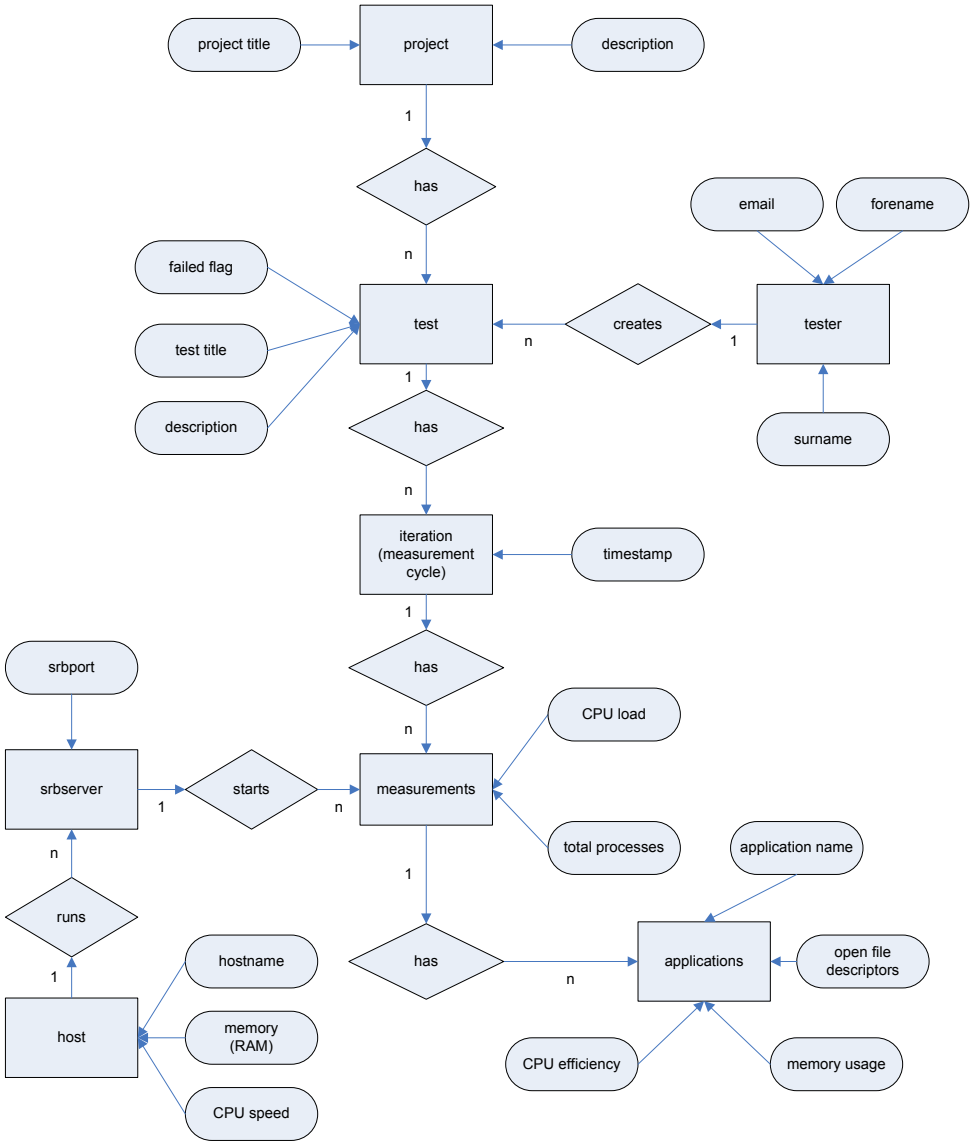


fig. 5.4: Entity Relationship Model

can find their measurements as fast as possible. The top-level entity is the project entity and every other entity is subordinated to it. Every project can have more than one test, but a test cannot be in more than one project. The tests will be created by a tester, which wants to do some measurements. The iteration entity stores the start of a measurement cycle and is connected to every measurement within a cycle. The measurement cycles represent the content of the diagrams, which will be displayed later in the GUI of the client application. A test stores the measurement cycles, which a tester has created. The measurement cycles run on one or more SRB servers. The SRB server entity runs on a host, which can have more than one of the SRB servers. The measurement entity stores the standard metrics, which depend on the machine and not on an application. The application entity stores the application specific data for one application. More than one application can be measured at one time. Therefore, the relationship between measurement and application is one to many.

5.2.2 Relational Data model - (RDM)

The Relational Data Model in Figure 5.5 shows the full database with all attributes, primary and foreign keys. The project table is identified through the project title, which is set

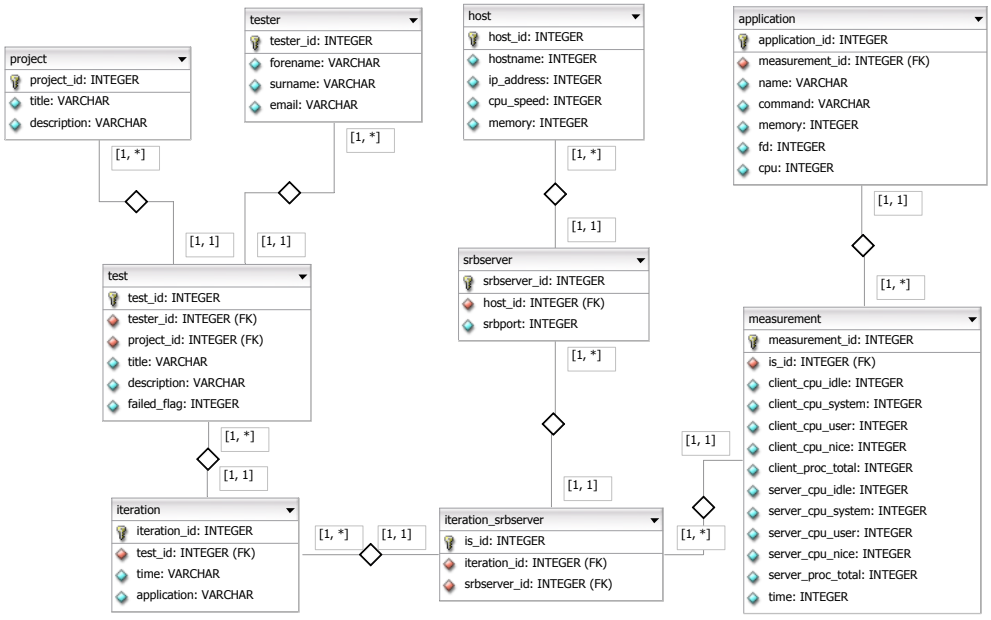


fig. 5.5: Relational Database Model

as unique. Therefore, every project name can be taken only once. As the name implies, the project description explains the motivation of the project. The test table has nearly the same attributes, whereas the test name is not unique, because similar test names are allowed, if the tests are part of different projects. The tester table needs the attributes surname and forename to be identified, so these attributes have to be different from "NULL". The email address might be interesting for other users. The iteration table, which consists of the measurement cycles, is defined by the starting time. This time presents the beginning of a measurement cycle.

The srbserver is identified through the srbport and the host_id, which is the primary key of the host table. The host table stores some standard information of a machine such as the hostname, which identifies the machine, the CPU speed, the amount of Random Access Memory (RAM) and the IP address of the machine.

To keep track of the database the two foreign keys srbserver_id and iteration_id have been outsourced to a join table iteration_srbserver. Join tables are used to solve many-to-many relationships, but in this particular situation it is just for a better overview and to disburden the measurement table. Every measurement can be allocated to a particular iteration and an SRB server. The is_id provides both srbserver_id and iteration_id. The measurement table consists of the metrics for the server machine and the client machine. The metrics are at the moment:

Table 5.2: Gmond metrics

Metric	Description
cpu_system	Percentage of the time the CPU ran system-level code
cpu_nice	Percentage of applications runs with the nice command
cpu_idle	Percentage CPU usage remaining while other processes are running
cpu_user	Percentage of the time the CPU ran user-level code
proc_total	Total number of processes

The measurement table is just a recommendation and might be changed depending on the values the user wants to measure. It is connected to the srbserver and iteration over the foreign key is_id. The applications, which have been measured on the server machine, are stored in an extra table application. The application table is connected over the foreign

key measurement_id to the measurement table. The measured values for an application are "fd" for open file descriptors, "cpu" for CPU usage, and "memory" for memory usage.

5.3 GUI Design

As already mentioned, the GUI consists of two dialogs. The first two sections explain the main dialog, which has two different frames. The third section explains dialog for the history diagrams and shows the different views.

5.3.1 Main Frame

The main frame is used to to configure a measurement and to browse through the tables, which are needed to create a diagram. The figure 5.6 shows listboxes, which contain the

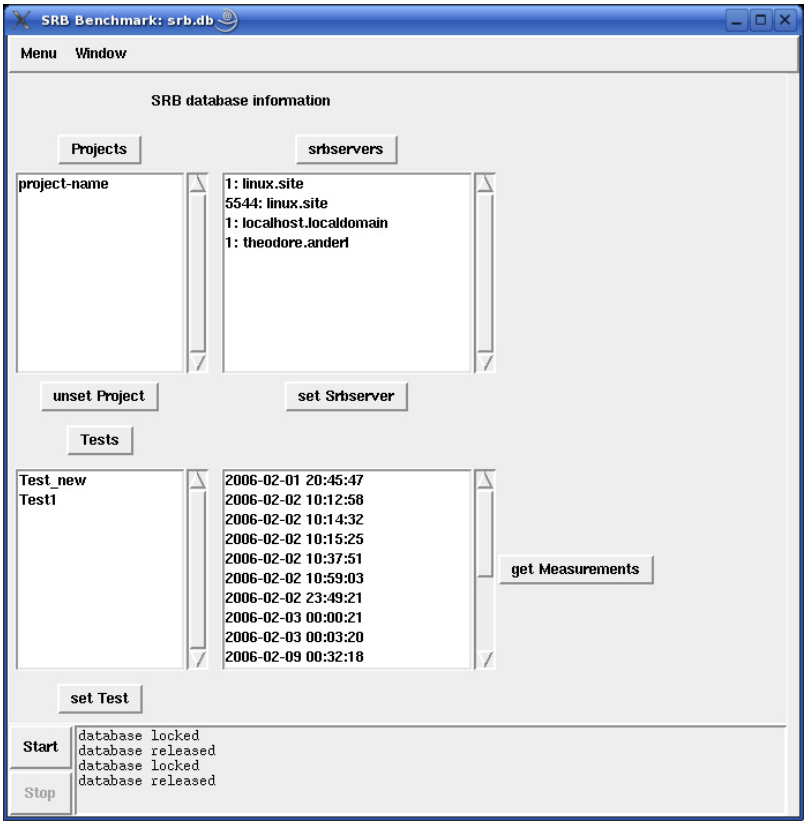


fig. 5.6: Main Frame

project, srbserver, test and iteration table. The SRB server entries begin with the SRB port followed by a colon and concluding with the hostname. For example, the entry “5544 : linux.site” is the SRB server on port 5544 on the linux.site machine. If the SRB server entry starts with a “1”, that means the machine was used for standard measurements without monitoring the SRB. The buttons below the listboxes can be used to set a particular value. At first only the project listbox and SRB server listbox are visible. The test table depends on the project and the srbserver table, which can refine the selection of tests and iterations. Afterwards the user can choose a test and the listbox for the iteration table will open with associated measurement cycles. If the user wants to see only diagrams of one SRB server, one should be set in the SRB server listbox. The iteration listbox is the only one, which allows a multiple selection to compare different measurement cycles of one test. The button “get measurements” will now produce one of the diagrams like in figure 5.8 for a single view or in figure 5.9 for a multiple view.

At the bottom of the dialog are the buttons to start and stop a measurement cycle. The text field next to the buttons shows the standard error output (stderr) and standard output (stdout) for verbose messages. The first menu “Menu” at the top of the dialog allows the user to open a different database file, to save the current database in another file, and to exit the programme. The second menu “Window” provides the possibility to change between the main and configuration frame.

5.3.2 Configuration Frame

The configuration frame has all parameters of the configuration file, which is used to configure the monitoring client. It provides a better view on the settings, which are necessary for a measurement cycle. The server listbox at the top is used to set up the number of servers, which shall be used for monitoring. For instance, if a user wants to set up two servers, he chooses "2" in the listbox and the parameter server.host and server.port will be available two times and can be configured. When the user has finished the setup, he has to confirm the settings with the set button. Furthermore, it is possible to write the configuration to a file by pressing the “Write” button. A small window will pop up and the user can change the name of the file or take the standard “config.ini”.

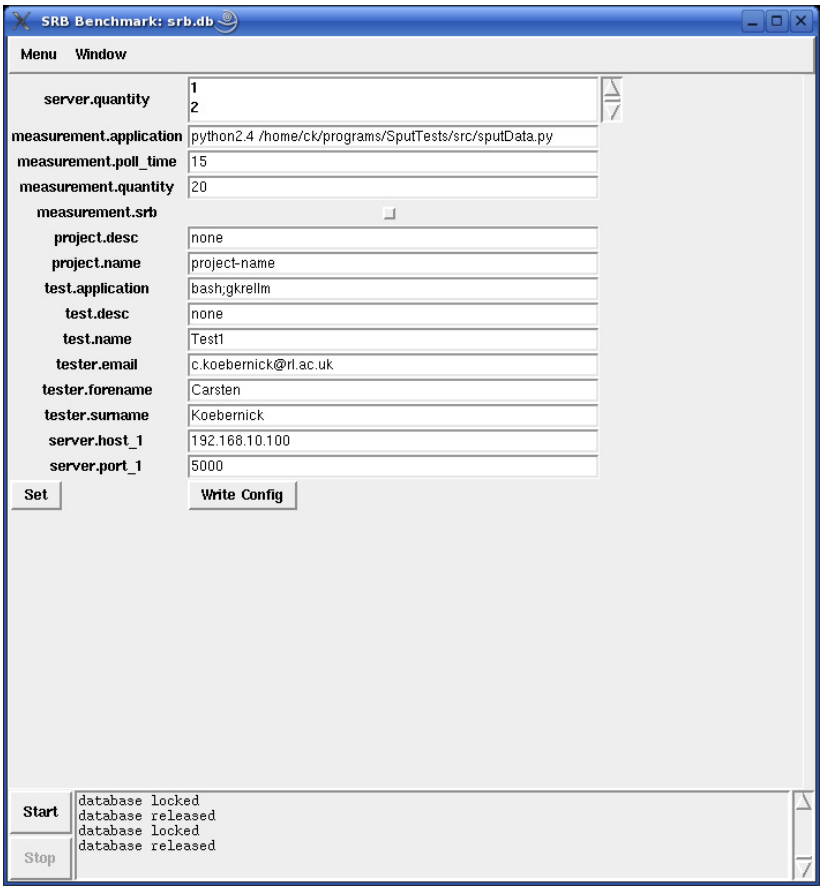


fig. 5.7: Configuration Frame

5.3.3 Diagrams

The diagram dialog shows the measurement graph in the upper frame. The x-axis is the time scale of the measurement cycle. The y-axis depends on the metric, which is shown. The memory and CPU usage values will be displayed in percentage. The listbox allows the user to switch between the different metrics. By double clicking in the listbox, a graph will be created for the new metric. The two sliders next to the listbox allow the zooming of the x-axis. The upper slider sets the center of the scale and the lower slider sets the number of values, which shall be shown. The zooming is also possible by using the left and right mouse button. A doubleclick on the left button zooms in on the current position and the right button zooms out. The “Save graph” button opens a little dialog, where the user can set a name for a postscript file to store the current graph. The button below the

"Save Graph" button creates a legend in the "Graph" frame, which makes it simple to recover a particular measurement cycle.

The first view in figure 5.8 is for a single measurement cycle on one SRB server. This

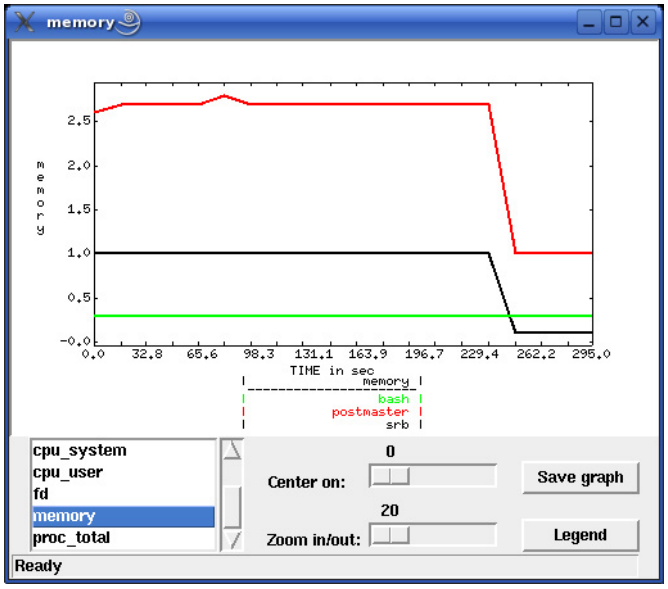


fig. 5.8: Single View Diagram

view can be used to compare the efficiency of different programs and the efficiency of the client and server machine. The figure shows the memory usage of three different applications. The "fd", "cpu" and "memory" links in the listbox are application metrics, whereas the "cpu_system", "cpu_user", "cpu_nice", "cpu_idle" and "proc_total" links are machine depended metrics and used to compare the server and the client with each other. The legend shows the for an application comparison, the name of the application and for a machine comparison it shows only the colour of the client graph and the server graph.

The multiple view in figure 5.9 on the following page is able to show multiple measurement cycles of one SRB server, one measurement cycle of many SRB servers, or multiple measurement cycles of many SRB servers. It allows the comparison of different machines or SRB servers. The application metrics are not in one graph anymore. The links are constructed by starting with the application name and ending with the metric name. For instance the application "bash" would have three links: "bash_fd", "bash_memory", "bash_cpu". Furthermore, the machine metric links have the same structure for the client

and the server values. For instance the “server_cpu_idle” compares all idle times of every different SRB server or measurement cycle. The legend shows for this view the SRB port, hostname and measurement cycle time.

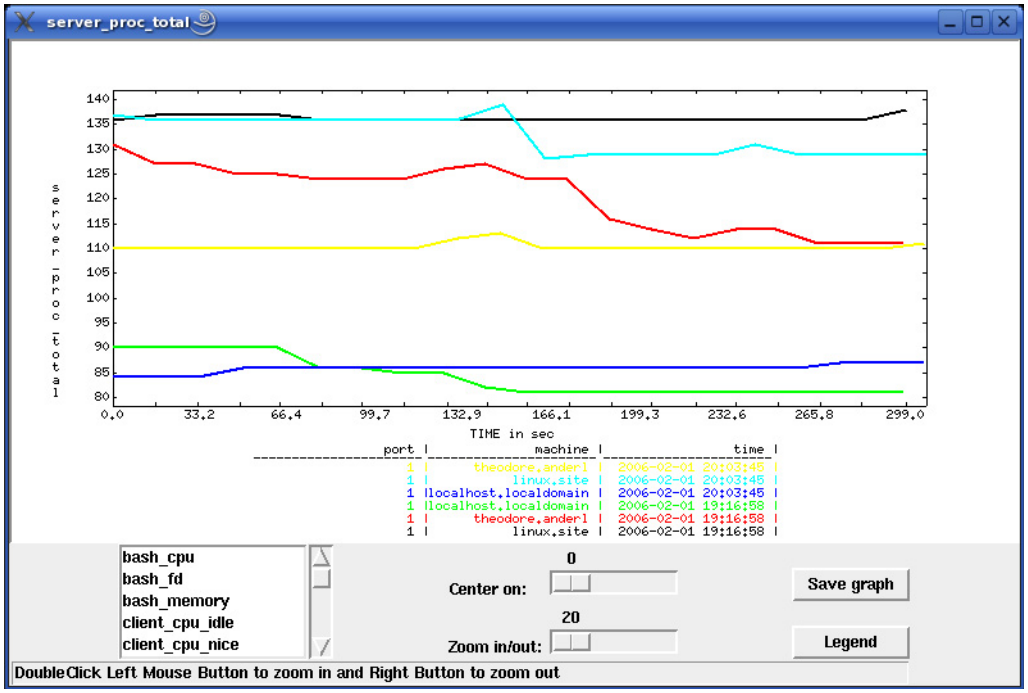


fig. 5.9: Multiple View Diagram

5.4 Modularisation

The monitoring client and the monitoring server have some similarities. Both use Ganglia and need a socket connection. So obviously, it makes sense to write modules, which can be used by both applications. Furthermore, the console client and the GUI client can use the same measurement functions and it would not again make any sense to write the functions twice. Therefore, the project has been divided into modules. A Python file is a module, which can be reused by importing the classes of the file in other modules. The project is divided into several modules to allow an easier extension of measurement system. The figure 5.10 on the next page shows the different modules of the system. The modules with the associated code are located in the Appendix B on page XXI. The

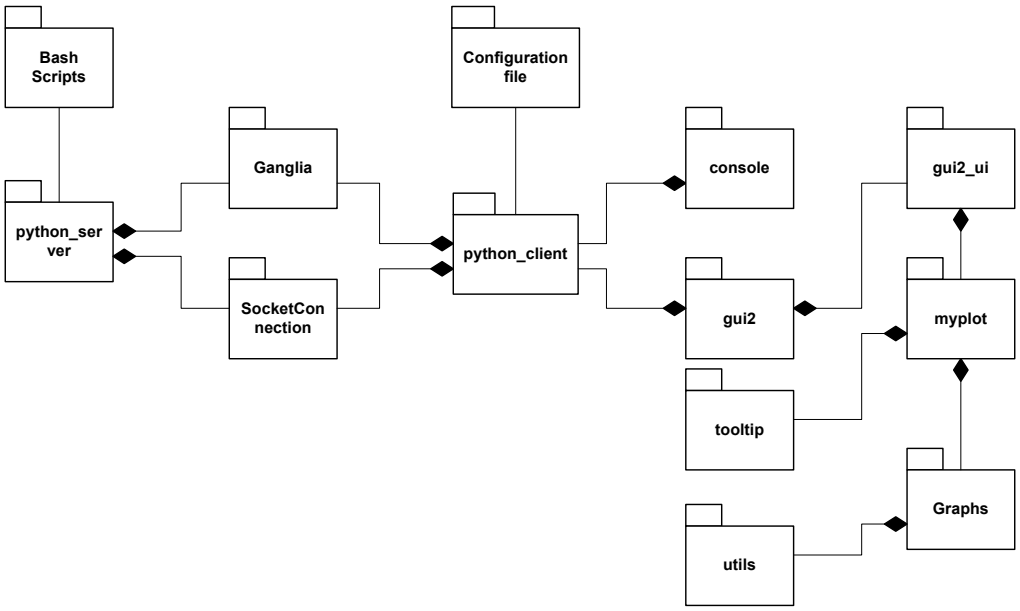


fig. 5.10: Modules

execution modules are “console” for the Console client, “gui2” for the GUI client and python_server for the server application.

python_server The server application uses the python_server, socket_connection and the ganglia modules. The socket_connection provides the server socket to connect that the client can connect to the server. The ganglia module provides all Ganglia functions, which means start, stop, and configure Ganglia. Furthermore the refreshment functions for gmond are included. The Bash scripts for the measurement of the SRB application are no modules, but stored in extra files. So single values can be measured without the measurement system.

console The console module uses the python_client module for measurement cycles and the database queries. The python_client module needs the socket_connection module and the ganglia module for the connection to the server and the own gmond daemon, which collects the measurement data of the other daemons.

gui2 The gui2 module uses similar to the console client the python_client module for measurements and database queries. But a lot more functions are needed to create the GUI. Therefore, the gui2_ui module provides the main dialog to browse through the tables and configure the measurement. It connects to the myplot module to draw

the history diagrams in another dialog. Finally, the drawing of the diagrams will be made by the Graph and utils module, which have been provided. Dr. Adil Hasan was so kind to provide these modules to finish the project in the given time period. The Graph Module has been edited to adapt the module to the measurement system. Besides that, the module has been extended with a legend function to describe the diagrams.

5.5 Object Models

The section will explain the modules in detail by describing the classes of the measurement system. The associated code is located in section B on page XXI in the appendix. All classes are displayed without the constructor function "__init__()" and the member variables. It would be to complex and for the monitoring client, it would exceed the space of a page.

5.5.1 Monitoring Server

The server consists of a "main" function and three classes, which are shown in figure 5.11. The main function reads at first the parameter from the command line with the "Cgetopt" class. The functions of the class are very trivial. The start function of the Cgetopt class gets the parameter and analyses them. It uses the standard module "getopt" from the Python library. Only three parameters are allowed, "-v" for verbose messages,"-h" for a little help output about the usage of the monitoring server, and "-p" to set up the connection port for the client. The "CSocketServer" class consists of all necessary functions to create, listen, read and write to a socket. After the parameter have been analysed the main function creates an object of the CSocketServer class to create a socket. Afterwards the "listen" function waits for a client connection. When the client connects to the server and sends the application names, the main function of the server starts the gmond daemon by calling the "start_gmond" function of the "Cinitialisation" class. In front of starting gmond the gmond.conf has to be adapted with the new IP address of the client. When gmond has been started, the main function will create an object of the "Refresh_Ganglia" class, which consists of a thread function that calls the "refresh_gmond" function. The Refresh_Ganglia class inherits from the Python standard class "threading.Thread" to use

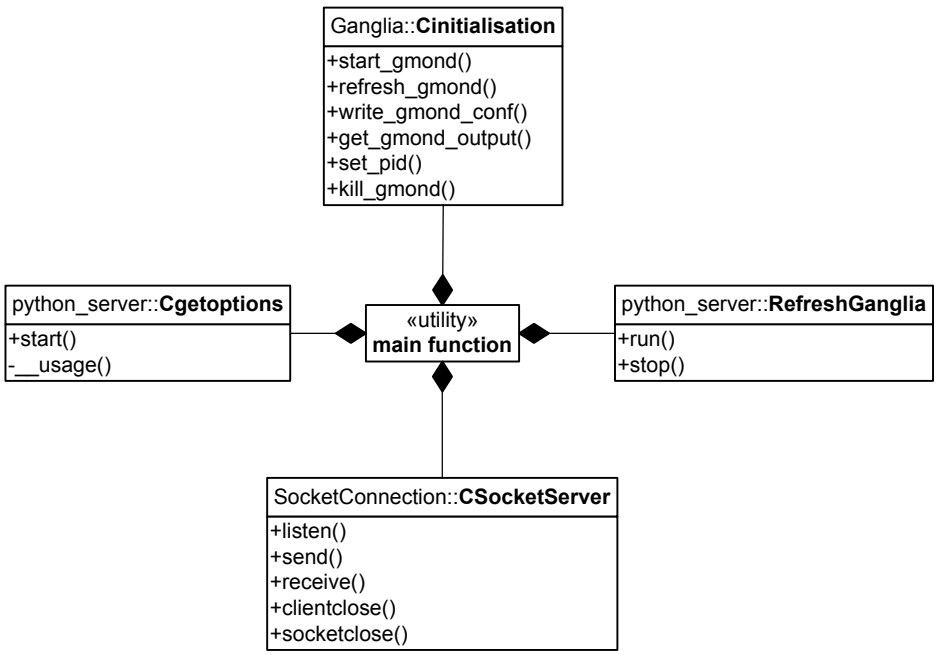


fig. 5.11: Server Object Model

thread functions. The refresh_gmond function monitors the applications with standard UNIX commands such as “ps” and “lsof”. If an SRB port has been sent before the application names, the refresh_gmond function will also use the three Bash Scripts called “average.sh”,”average_mem.sh” and “num_fd.sh” to measure the CPU efficiency, memory usage and number of open file descriptors of the SRB server applications. The Bash scripts are attached in the appendix in chapter C on page XC.

The return values of the Bash scripts will be passed with gmetric to update the measurements for the client. The main function of the monitoring server waits for a “clientstop”-string to stop the Refresh_Ganglia thread, stop gmond with the “kill_gmond” function, and to close the socket with the function “clientclose”. The “set_pid” function stores the process ID of the gmond daemon. The kill_gmond function will need it to stop the gmond after the measurement cycle has been finished. Afterwards the main function waits again for a new connection from a client. When server has been stopped by pressing “CTRL+C”, everything will be closed and the socket will be stopped by using the “socketclose” function.

5.5.2 Monitoring Client

Caused by the complexity of the client and its modules, the class diagrams are separated into two parts. The first part consists of the client functionalities and the console application.

5.5.2.1 Operations and Console

The figure 5.12 on page 61 shows all classes of the python_client and the console module. The console module consists only of one class, which provides the tables and gets the parameter from the command line. The start function of the "Cnogui" class filters the parameters and catches incorrect inputs. The following parameters are allowed for the console client:

Table 5.3: Console Client Parameter

Parameter	**Exercise**
primary-keys	
-p, –project-id <id>	passing a project id
-t, –test-id <id>	passing a test id
-i, –iteration-id <id>	passing an iteration id
-s, –srbserver-id <id>	pass a srbserver id
tables	
–projectlist	draw a project-table
–testlist	draw a test-table
–iterationlist	draw an iteration table
–srbserverlist	draw a srbserver table (including host parameter)
–measurementlist	draw a measurement table
other parameter	
-s, –startday <YYYY-MM-DD>	used in combination with endday to get measurements between two dates
-e, –endday <YYYY-MM-DD>	
-d, –database <file>	pass a database (standard:srb.db)
-c, –config-file <file>	pass a config file (standard: config.ini)

Continued on next page

Parameter	Exercise
-f, –config-dump <file>	store an config example in a new config file
-g, –gauge	start a measurement
-h, –h, –help	draw the usage of the client
-v, –verbose	draw more messages during the process of the client application

The list parameters start the associated list functions, for example the parameter "–testlist" starts the "__test_list" function. These functions create the command line table. Only the project table is included in the "__console__" function. The primary keys refine the tables to see only particular information. The "-g" parameter starts the measurement by starting the "__measure" function. Furthermore, a configuration file can be passed with the "-c" parameter and the configuration file can be displayed with "-f ".

The Cnogui class creates at first an object of the "Cconfigparser" class to parse the configuration file. Therefore three functions are needed. The "load_config" function loads a particular configuration, the "write" function writes and the "get_config_file" function returns the name of the current configuration file. The second class is the "CMeasurement" class. It connects the main classes, which are needed for a measurement cycle, together. The function "__get_srbport" of the CMeasurement class parses the ".MdasEnv"-file to get the SRB port of the running SRB system. The constructor of the CMeasurement class creates at first an object of the "Cdatabase" class to provide all other classes with the same database pointer. The "run_measurement" function starts a measurement cycle and creates a "ServerConnect" object to connect and assure the connection to the monitoring servers. This class uses the "CSocketClient " class of the "socket_connection" module to listen, receive, send and stop the server. After the connection has been established, an object of the "GangliaThread" class will be created to finally start the measurement cycle. The Thread will start its own gmond daemon with the Cinitialisation class and parse the frequently measured output of the gmond daemon with the "Cxmlparser" class. The parser will be explained with more details in section 6.3 on page 73. The "Crunapp" class starts the test application, which should run during the measurement cycle and to create load on the SRB system. The application will be started as a thread and will run until the measurement cycle has stopped. If the test application is not finished with its task, it will be "killed" by the monitoring client.

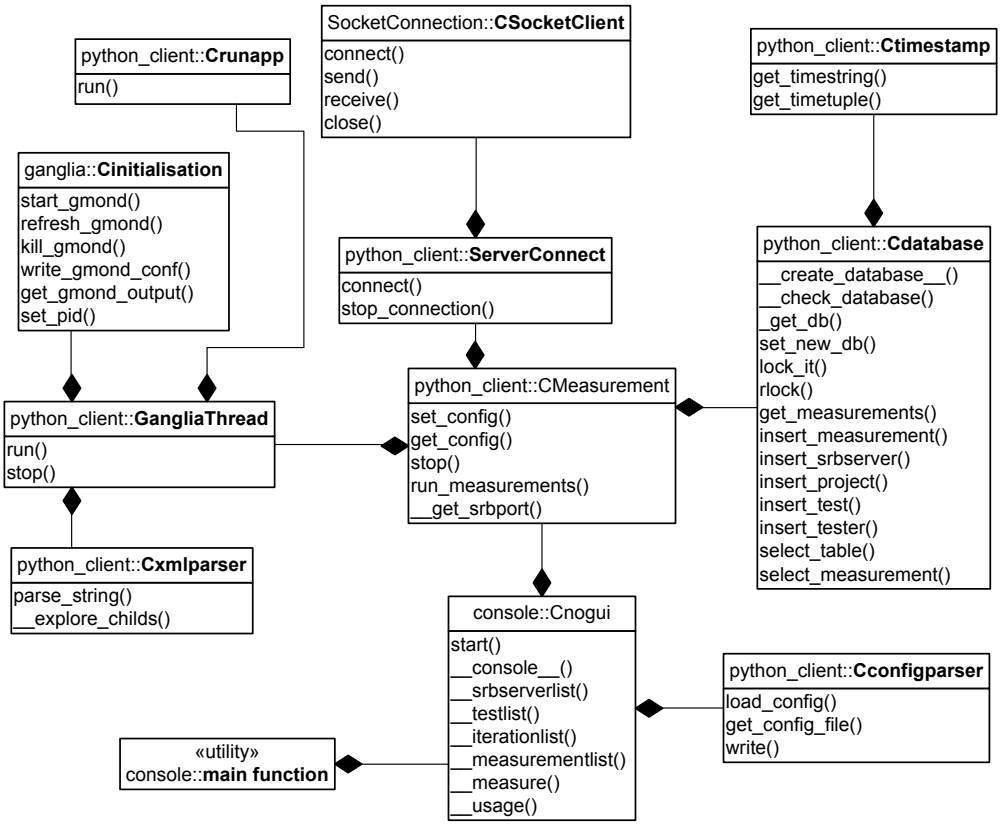


fig. 5.12: Client Object Model

The run_measurement function is finished after the GangliaThread object has been created and the “run” function has been started with the “start” command. That is the usual way to start a thread function of a threading class. It is similar to Java threading. The CMeasurement class can stop the GangliaThread with the “stop” function. The “set_config” and “get_config” function are used to handle the configuration between the file and the classes.

The Cdatabase class provides all necessary database functions. If a client has been started and no database file is available, the “__create_database__” function creates a new “srb.db” database file with all necessary tables. The “__check_database__” function verifies the given database, whether the file is broken or not. The “set_db” and “get_db” functions set and get a new database file. All different insert functions insert a new row in the table and return the current primary key of the row. As mentioned in the SQLite section 3.6.4, it is not advisable to use one database pointer in different threads, but with more than one database pointer write locks may happen, if more than one database pointer

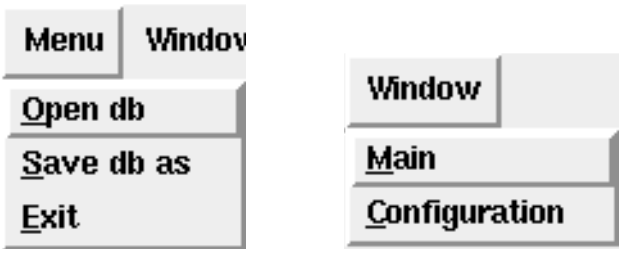
wants to write in the database at the same time. Therefore only one database pointer will be used, which has been created in the Cdatabase class. The functions “lock_it” and “rlock” are threading functions, which lock the database before a query, and release it afterwards. This “Mutual exclusion” (Mutex) or binary semaphore allows the threads to use one database pointer. A Mutex is used to synchronise processes and restrict them to use a special part of the code at the same time. This avoids inconsistent states. Furthermore, the Cdatabase class uses a “Ctimestamp” object to create a time string, which sorts the measurements and measurement cycles by time.

5.5.2.2 GUI classes

The classes of the Graphical User interface are shown in figure 5.14 on page 64. These classes use the Tkinter library to create the GUI. The two dialogs are separated in two modules. The complete main dialog of the GUI is stored in the gui2_ui module and the diagram dialog is stored in the myplot module.

The gui2 module starts the GUI with “Cguistart” class. The “start” function parses the arguments passed by the user, which are the same as in the console client, but without any parameters to create a table in the command line. The “usage” function displays a help for the GUI client. The “CPrintinGUI” class overwrites the “write” function of the stderr and stdout descriptor. The overwritten write function of the descriptors passes the messages to the text widget of the main frame. The “stop” function disables the passing and recreates original write function.

The “Dialog” class of the gui2_ui module is the top-level class, which creates the main dialog. As the names imply, the “button_start_command” and “button_stop_command” can start and stop a measurement cycle. The same applies to the “open_db” and “save_db” functions, which can open a database file and save a current database under a different name. The “_test_stop” function waits for the end of a measurement cycle. If the maximum number of measurements has been reached, the function enables the “Start” button and disables the “Stop” button. The function runs in an additional thread, which only runs when a measurement cycle is running. The “_change_text_in_console” function is used for the CPrintinGUI class of the gui module to finally write the output of the stdout and stderr into the text widget of the main dialog. The last function of the Dialog class is the “_gui_quit” function, which stops the GUI and closes every open diagram dialog.



(a) "Menu" (b) "Window"

fig. 5.13: Menus

The "Mymenu" class provides the menus for the main dialog, which are shown in figure 5.13. The first menu is created by the "__create_menu_1" function, which connects the GUI to the _open_db, _save_db and the _gui_quit function. The "__create_menu_2" function provides the functionality of the "Window" menu. It is used to change between the "graph_choice" class and the "Configuration" class, which both provide a different frame for the main dialog. Both frame classes use the "Mainframe" class. The frame that will be created in the Mainframe class can be scrolled, because it consists of a Tkinter "Canvas" widget, which is normally used as a draw environment, with an embedded "Frame" widget. Therefore, the graph_choice and the Configuration classes are developed on top of that scrollable frame. The "canvas_reload" function has to be executed, whenever a frame has been changed to update the dimensions. The Mainframe class uses the "AutoScrollbar" class, because it creates scrollbars at the left and lower side of the frame, but only when they are needed and an inner widget of the frame ran out of space. Furthermore, the text widget at the bottom of the GUI uses the automatic scrollbars as well.

The graph_choice class provides the main frame for the main dialog. It allows to browse through the tables and to choose a particular measurement cycle, which should used to generate a history diagram. The "config" function creates the main frame with the first two list boxes project and SRB server. The list boxes will be created with Tkinter widget "Listbox". The "__set_test" function creates the test list box and the "__set_test_command" function creates the iteration list box. The functions "srb_server_table", "__test_table", "__project_table", "__srbserver_table" create an overview of the table content by drawing simple GUI table. These functions are obsolete, but have been used during the implementation of the measurement system and might help for a further development. The overview tables are still available in the GUI. By pressing a button over one of the list boxes, the

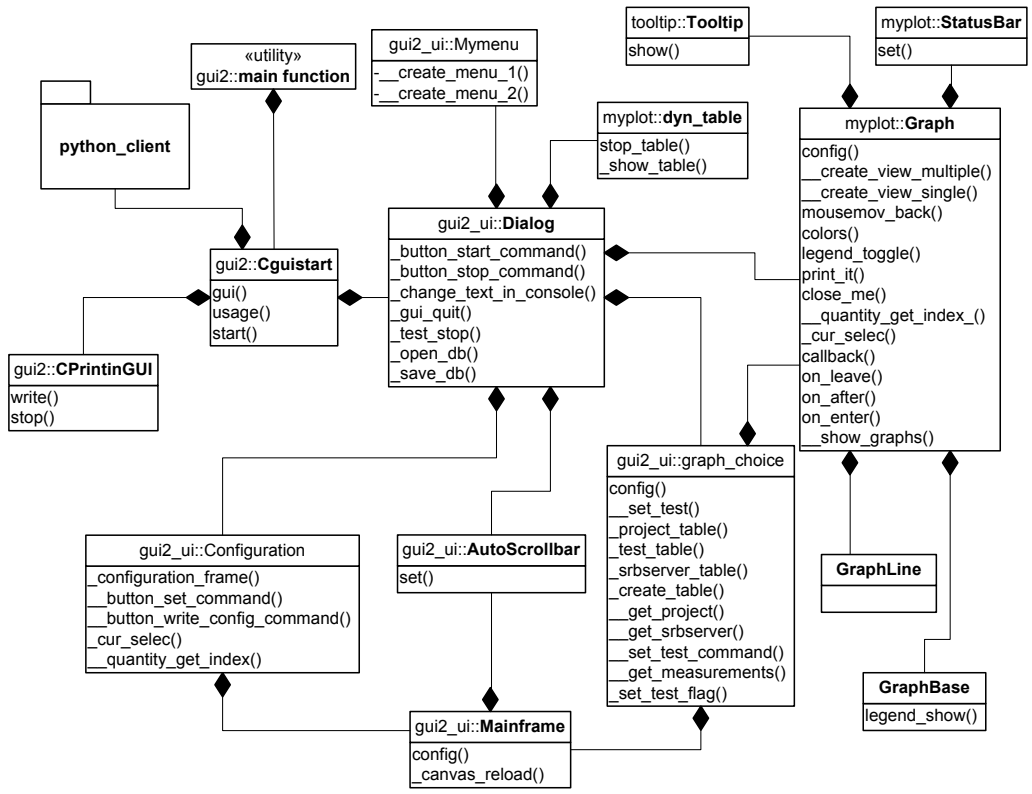


fig. 5.14: GUI Object Model

particular table will be created. The “get_” functions gather the content of a list box, if one of the “Set ...” buttons below a list box has been pressed.

The Configuration class contains five functions. The “_configuration_frame” function and the constructor create all necessary “Label” and “Text” widgets to show the content of the configuration file. The “Set” button of the configuration frame uses the “_button_set_command”, which configures the measurement cycle with the current information from the frame. The “_button_write_config_command” writes the configuration to the current or to a new file. The last two functions “_cur_selec” and “__quantity_get_index” are used to get the number of the “server.quantity” list box.

The myplot module is the second GUI module, which contains the classes of the diagram dialog. The “Graph” class creates the dialog with all needed widgets. The main task of the Graph class is to create the two particular views. The “_create_view_single” and “_create_view_multiple” are responsible for the lists, which are necessary to draw the diagrams. Both functions get the content of the measurement table from the python_client

module and have to prepare the data for the “Graphs” module. The “show_graphs” function finally passes the lists to the Graphs module. The “config” function decides, which view function must be used and connects the functions with the widgets. The “callback” function is responsible for the x-axis zooming within the diagram. The “on_enter”, “on_after”, and “on_leave” function create, show and destroy a tool tip with the “Tooltip” class and change the content of the status bar with “StatusBar” class. Both classes are used to provide auxiliary information about the widgets of the Graph class. A tool tip will be displayed, when the mouse has been moved over a widget. The mouse movement will be recognized with the “mouse_mov” function, which uses the Tkinter “Event” widget. The “colors” function assigns a colour to a graph. The legend also uses the colour to connect a single row with a particular graph. The legend will be drawn by the “legend_show” function in the GraphBase class, which is part of the Graphs module. The “legend_toggle” function starts and stops the legend and is connected with the “Legend” button in the diagram dialog. The “Save graph” button executes the “print_it” function, which opens a small dialog to choose the target for a postscript file, which contains the current graph. The last function “_close_me” of the myplot module destroys the dialog.

The “dyn_table” class is used to start the dynamic graph for a running measurement cycle. The “_show_table” function will be started as thread by the _button_start_command function of the Dialog class. The function gets every second new measurements from the database and passes them to a Graph object from the myplot module. Therefore, whenever new measurements arrive the diagram dialog will be redrawn. The “stop_table” function destroys the diagram dialog and stops the thread.

Chapter 6

Implementation

The following sections describe five code examples in detail. These sections deal with the socket connection, the measurement of the applications, the parsing of the Ganglia XML output, and the preparation of a list for a multiple view in the diagram dialog.

6.1 Socket Connection

A socket connection has been used between the monitoring server and the client. This enables the client to configure the server remotely. The Python socket module provides the funcionality and is very easy to use and is similar to the C socket connection. The socket module will be imported with the command `import socket`. The "Csocketserver" and "Csocketclient" classes use their constructor to create a TCP socket. The listing 6.1 shows a code snippet of the constructor with the three main commands.

Listing 6.1: Create Socket

```
def __init__(self)
    self.p_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.p_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1);
    self.p_sock.bind((socket.gethostname(), self.i_port))
```

The first parameter "socket.AF_INET" of the socket function indicates that the socket uses the Internet Protocol (IP). The second parameter "socket.SOCK_STREAM" indicates that a stream should be send over socket. If nothing else has been set, the socket uses TCP. The UDP parameter would be "socket.SOCK_DGRAM". The "setsockopt" function is not necessary, but useful if a socket is stuck while it is connected to another

host. The function allows reusing the socket, even though it is set as engaged. But the setsockopt function has many other functionalities, which are not used in this command, but can be refered in the UNIX man pages [21]. The bind function applies only to the server, because a server socket has to be bound to a local address. The bind function needs a "tupel" as parameter, which is a collection of other objects and these objects are not changeable. [22]. The first parameter in the "tupel" is the hostname and the second parameter opens the port for remote connections.

Listing 6.2: Listen function

```
def listen(self):
    self.p_sock.listen(1)
    self.p_conn, i_remote_host = self.p_sock.accept()
    return i_remote_host[0]
```

The listing 6.2 shows the listen function of the Csocketserver class, which waits for a connection from a client. The listen function of the socket module takes one parameter, which indicates how much connections the server handles at one time. Finally when a client tries to connect, the accept function establishes the connection.

Listing 6.3: Connect function

```
def connect(self, i_remote_host = '', i_remote_port = 0):
    self.p_sock.connect((str(self.i_remote_host), int(self.i_remote_port)))
    return 0
```

The connect function in listing 6.3 is used by the Csocketclient class to connect to the server. The parameters are the remote host and port of the server. The host parameter can be an IP address or a host name.

The “send” and “receive” function in listing 6.4 are similar in the Csocketclient and Csocketserver class. Before sending the data from the client to the server, it is necessary to get the string length of the data. It is necessary to send the length, because if the send function has been executed several times in series and different data packages have been send, it is possible that the receive function get more than one package at one time and it would not be possible to assign the data to its task. By converting the string length to network byte order, the code is kept independent from the machine architecture. There are two different versions of byte orders, Big-endian (IBM mainframes) and Little-endian (most PC’s). Therefore, the "htonl" command converts the integer in Big-endian format, which is the same as the network order, to unify the byte order. The receive function on

the client side converts it back to an integer with the "ntohl" function. The pack function of the "struct" module has to wrap the number again, because the send function normally sends only strings. The function converts the Big-endian integer into a binary string. The first parameter “L” indicates an unsigned long integer. On a normal PC architecture the unsigned long is four Bytes, which means the number can be up to 2^{32}.

Listing 6.4: Send & Receive function

```
def send(self, data):
    s_content = str(data)
    n_length= socket.htonl(len(s_content))
    size = struct.pack("L", n_length)
    sent = self.p_sock.send(size+s_content)
    return 0

def receive(self):
    size = struct.calcsize("L")
    size = self.p_conn.recv(size)
    size = socket.ntohl(struct.unpack("L", size)[0])

    data_send = data = ""
    size_decrement = size

    while len(data_send) < size:
       if size_decrement < 1024:
           receive_size = size_decrement
       else:
           receive_size = 1024
           size_decrement -= 1024
       data = self.p_conn.recv(receive_size)
       data_send = data_send+data
    return data_send
```

The send function transmits the string length and the data at the same time. The receive function gets the first four Bytes to know the length of the following string. The length of the data will be unwrapped and unpacked as already described and afterwards the while loop of the receive function gets the data in 1024 Byte slices and connects the partial strings to one string together.

6.2 Application Measurements

6.2.1 Standard Application

The measurement of standard application calculates three particular values, the CPU, the memory usage, and the number of open file descriptors. To measure the three new metrics, the monitoring server needs the application names from the client. The names were passed in a string, where a semicolon divides the applications. The “refresh_gmond” function of the “Cinitialistation” class in the ganglia module parses the string, monitors the applications and passes the results with gmetric to gmond.

The main part for the standard application measurement is listed in 6.5 on the next page. Before the application can be measured, the string with the application names from the client has to be split and stored in a list. Afterwards the “for” loop uses the list as a parameter to do the measurements individually for every application. The line 14 shows the first bash command, which gets the data for the memory and the CPU usage at once. The command returns the measurements in the following way:

```
1.0 2.0
3.0 1.0
1.3 2.1
...
```

Every row of the return string from the bash consists of the memory and CPU usage for one process of the application. Therefore, all values will be separated and the memory usages will be added to one value and CPU usages to another one. The separation and summation will be done in the lines 15 till 22.

Calculating the number of open file descriptors is a bit different. At first, it is necessary to gather all running Process IDs (PID) of an application. This will be done with bash command in line 37. The next step is using a “for” loop to get all open file descriptors of every PID. The line 45 contains the bash command, which counts the lines of the “fd” file in the “/proc/<PID>/” directory. Every line in this file indicates an open file descriptor with exception of the heading. Therefore, the line 46 adds the number of open file descriptors together and subtracts one line for the heading. When the "for" loop has

finished with all PIDs, the number of file descriptors is certain. Now all three metrics for one application are finished and can be passed with gmetric to gmond.

The new metrics for the gmond daemon need a name with which one can regain them. Therefore, the new metric name consists of the application name and the meaning of the measurement. For instance, the CPU usage of the “bash” application would be stored in the metric “bash_cpu”. The XML snippet below shows the content of a “METRIC” tag for the “bash_cpu” metric.

```
<METRIC NAME="bash_cpu" VAL="0.1" TYPE="float" UNITS="Byte" TN="0" TMAX="60" DMAX="0"
    SLOPE="both" SOURCE="gmetric" / >
```

The three gmetric commands, which have to be executed to pass the data to gmond, will be connected to one string and executed with one “os.system”-command. This command finally executes the three gmetric commands in line 69.

Listing 6.5: Server application measurement

```
1  ...
2  if application != '':
3      apps = application.split(';')
4      if apps[-1]=='':
5                  # delete the last member of the list if it is empty
6          del apps[-1]
7
8      # delete the first empty spaces of the command
9      p = re.compile('^ ')
10     s_app = []
11     for i in range(0,len(apps)):
12         apps[i] = p.sub("",apps[i])
13         f_cpu = f_mem = 0.0
14         s_cpu_mem = commands.getoutput("""ps -C "%s" -o pmem,pcpu | awk '{print $1 " "
               $2}'""" % apps[i])
15         if s_cpu_mem != "":
16             cpu_mem = s_cpu_mem.split("\n")
17
18             for x in cpu_mem:
19                 if re.match("[0-9.]",x):
20                     x = x.split(" ")
21                     f_cpu += float(x[1])
22                     f_mem += float(x[0])
23
24         s = re.compile('[a-zA-Z0-9_-]+')
25         s_app.append(s.match(apps[i]))
26         s_app[i] = s_app[i].group()
27         i_count =0
28         for x in range(0,i):
29             if s_app[x].find(s_app[i]) != -1:
30                 i_count +=1
```

```
31          if i_count != 0:
32              print "not two commands of the same application"
33              continue
34
35          """ set the used file_descriptor value of the srb_server """
36          l_pid = []
37          s_pid = commands.getoutput("ps u -C '"+apps[i]+"' | grep $USER | awk '{print $2
                }'");
38
39          #""" just the open application of the user will be measured"""
40          l_pid = s_pid.split("\n");
41          #""" divide the different application pids"""
42          i_number_fd = 0
43          for s_single_pid in l_pid[0:]:
44              try:
45                  s_number_fd = commands.getoutput("ls -l /proc/"+s_single_pid+"/fd | wc -
                        l")
46                  """ get the number of open file descriptors"""
47                  i_number_fd += int(s_number_fd)-1
48                  """ the header should not be counted """
49              except ValueError:
50                  print "Programme %s cannot be monitored!" % apps[i]
51                  break
52
53          stringer = self.ganglia_folder+"""bin/gmetric -c """ \
54          +self.ganglia_folder+"""sbin/gmond_test.conf --name  """+s_app[i]+"""_mem \
55          --value "%s" --type float --units Byte""" % f_mem
56
57          stringer = stringer+";"+self.ganglia_folder+"bin/gmetric -c " \
58          +self.ganglia_folder+"sbin/gmond_test.conf --name "+s_app[i]+"_fd --type int16
                \
59          --value "+str(i_number_fd)
60
61          stringer = stringer+";"+self.ganglia_folder+"""bin/gmetric -c """ \
62          +self.ganglia_folder+"""sbin/gmond_test.conf --name """+s_app[i]+"""_cpu \
63          --value "%s" --type float --units Byte""" % f_cpu
64
65          stringer = stringer+";"+self.ganglia_folder+"""bin/gmetric -c """ \
66          +self.ganglia_folder+"""sbin/gmond_test.conf --name """+s_app[i]+"""_cmd \
67          --value "%s" --type float --units Byte""" % apps[i]
68
69          os.system(stringer)
70  ...
```

6.2.2 SRB Application

The measurement of the SRB application is slightly different. The “srbMaster” process of an SRB system creates for a new query to the SRB system a new process. Therefore,

it is necessary to get the PID of the srbMaster process. The srbMaster is the Parent PID (PPID) of all other SRB server processes of one SRB system. Whereas more than one SRB system could run on one machine, it is necessary to find the correct srbMaster process. The line 7 of the listing 6.6 uses the Linux tool "lsof" to get the srbMaster process of the SRB system to which the monitoring client belongs. The lsof tool shows all open descriptors of the current machine. By passing the srbMaster process with the parameter "-c" to lsof, the output will be limited. Now, it is possible to pipe the output to the"grep" command and search for the SRB port of the SRB system. The pipe means, that the standard output (stdout) of lsof will be used for the standard input (stdin) of the grep command. "awk" will finally get the PID of the srbMaster process from the stdout of grep.

The PID is used to get the CPU efficiency of the SRB server processes. Line 15 executes "ps" and gets the CPU usage of all processes, which have the PID of the srbMaster process as PPID. Afterwards the for loop in line 18 adds all CPU usages together and the sum is the return value for gmetric.

Listing 6.6: CPU usage of the SRB server

```
1  if ! [ $1 ]
2  then
3      exit -1;
4  else
5      var=$1
6  fi
7  ppid=$(lsof -c srbMaster | grep -i $var | awk '{print $2}' )
8  #| sed -e 's/*://')
9  if ! [ $ppid ]
10 then
11     echo "0"
12     echo "no srbMaster found"
13     exit 0
14 fi
15 list=(`ps -o pcpu --ppid $ppid`)
16 sum=0.0
17 list_length=${#list[@]}
18 for ((i=$list_length-1;i>0;i--))
19 do
20     sum=`echo ${list[${i}]}+$sum | bc`
21 done
22 echo $sum
23 exit 0
```

6.3 XML Parser

The XML parser consists of two main functions, where the first, parse_string gets the XML string and parses it with the Python xml.dom.mindom module. The “parseString” function of the inbuilt module returns a particular object in line 7 of listing 6.7. This object needs to be converted for a further use.

Listing 6.7: “parse_string function”

```
1 def parse_string(self, s_xml):
2     """ parse the string start the xml-string to dictionary function"""
3     self.__d_xmlstream = {}
4     self.__s_xml = s_xml
5     if self.__i_verbose:
6         print "XML-String:\n%s" % (s_xml)
7     self.__doc = xml.dom.minidom.parseString(self.__s_xml)
8     self.__explore_childs()
9     return self.__d_xmlstream
```

The second function “__explore_childs” in listing 6.8 on the next page will be executed by the “parse_string” function to convert the object into a dictionary. This function is recursive, which means it starts itself again and again. It opens the nested XML output and gathers the important attributes. For converting the XML output, it is necessary to have a closer look on the gmond XML structure:

```
1  <GANGLIA_XML>
2  <HOST NAME=<hostname> IP=<IP address> REPORTED=<Time of Transfer> ...>
3  <METRIC VAL=<value> TYPE=<type> UNITS=<units> ...>
4  </METRIC>
5  <METRIC ...>
6  ...
7  </HOST ...>
8  <HOST>
9  </HOST>
10 </GANGLIA_XML>
```

The first important tag is the “HOST”, which surrounds all measurements of one machine. Therefore, it is the first key of the dictionary. The second key are the attributes of the metric tag. Five different keys are possible. The “VAL” attribute is the only one that is important for storing in the database, but the “TYPE” and “UNITS” attributes have been parsed as well between line 18 till 21. This might be useful for a future use, with a different database model. Furthermore the “REPORTED” and “IP” attributes of the “HOST” tag are second keys, because the “REPORTED” attribute contains the time of

the last added metrics, which helps to find out if the values of the XML output are up to date and the server on the other side still passes new measurements with gmetric into gmond. The "IP" attribute is the IP address of the host and will be stored for every new machine in the host table of the database.

Listing 6.8: XML convert function

```
def __explore_childs(self):
    """ get the xml_stream in a dictionary (recursive function)"""

    nodelist = self.__doc.childNodes  #get the childs of a tag
    for subnode in nodelist:
        if (subnode.nodeType == subnode.ELEMENT_NODE):

            if (subnode.tagName == "HOST"):
                # create the first key of the dictionary
                self.host = subnode.getAttribute("NAME")
                self.__d_xmlstream[self.host] = {}
                self.__d_xmlstream[self.host]["IP"] = subnode.getAttribute("IP")
                self.__d_xmlstream[self.host]["REPORTED"] = \
                 subnode.getAttribute("REPORTED")
            if (subnode.tagName == "METRIC"):
                  # save the VAL, TYPE and UNITS attribute in the dictionary
                self.__d_xmlstream[self.host][subnode.getAttribute("NAME")] = {}
                self.__d_xmlstream[self.host][subnode.getAttribute("NAME")]['VAL'] = \
                 subnode.getAttribute("VAL")
                self.__d_xmlstream[self.host][subnode.getAttribute("NAME")]['TYPE'] = \
                 subnode.getAttribute("TYPE")
                self.__d_xmlstream[self.host][subnode.getAttribute("NAME")]['UNITS'] = \
                 subnode.getAttribute("UNITS")
            self.__doc = subnode
            self.__explore_childs()
```

6.4 Client Measurement Thread

The Measurement Thread needs the XML parser of the last section to parse data during a measurement cycle. The use of a thread offers the possibility to interact with the GUI during a measurement cycle. One way to create a thread in Python is using the threading module. The GangliaThread class inherits from the Thread class of the threading module:

Listing 6.9: GangliaThread constructor

```
class GangliaThread(threading.Thread):
    def __init__(self, ...):
              threading.Thread.__init__(self, name = "Refresh_gmond")
              ...
```

The next step is starting the constructor of the superclass to enable the thread functionality for the GangliaThread class. Afterwards it is possible to create a run function, which will run as a thread. The “start” command will execute the run function.

The Thread, which is in fact the run function, is shown in listing 6.10. The function starts at the first the gmond daemon with start_gmond function of the Cinitialisation class. Afterwards gmond needs a little time to configure and provide all the data. Thus in line four, the thread sleeps ten seconds to give Ganglia some time. The "while" loop in line ten starts the measurement cycle by querying the local gmond with telnet. The XML output can be parsed after the XML string has been cut out of the telnet output.

The next step is to write the data into the database. The "for"-loop in line 17 verifies that every monitoring server is represented in the XML data. Afterwards the line 22 checks if a particular server has sent its data. If all measurements have been sent, the Thread tries to find the SRB server in the database. If the machine is not available, a new entry will be created. If the remote computer has sent all necessary measurements and machine information, the line 36 creates a new row in the join table "iteration_srbserver". Afterwards, the line 39 verifies that the Ganglia output is up to date. For example, if the "REPORTED" attribute from the "HOST" tag has the same value as the “REPORTED” attribute from the last query, the measurements are old and an error message will be returned. The "srbserver_id" of the iteration_srbserver table will be used in line 43 to finally insert the current measurement in the "measurement" and "application" table. If every remote gmond has provided its measurements, the run function will start in line 49 an extra thread, which starts the "test_application". The "while" loop will be closed, if the maximum number of measurements has been reached or the user has pressed the “Stop” button, which will be recognised through the "_stopevent" variable. This variable will be set, if the user wants to stop the measurement cycle, manually.

Listing 6.10: run function of “GangliaThread” class

```
1  def run(self):
2      """ Thread which polls Ganglia"""
3      self.__ganglia.start_gmond(gmond_conf="gmond_client.conf")
4      time.sleep(10)
5      application_started = 0
6      counter = 0
7      srbserverid_set = {}
8      last_insert = {}
9
10     while not self._stopevent.isSet():
11         s_xml_content = self.__ganglia.get_gmond_output()
```

```
12          i_xml_start = s_xml_content.find("<GANGLIA_XML");
13          i_xml_end = s_xml_content.find("</GANGLIA_XML>")+14;
14          d_xml = self.__parse_object.parse_string(s_xml_content[i_xml_start:i_xml_end])
15          host_count =0
16
17          for machine in d_xml.keys():
18              if machine == 'localhost':
19                  continue
20              if not srbserverid_set.has_key(machine):
21
22                  if d_xml[machine].has_key("cpu_speed") and d_xml[machine].has_key("
                        hostname"):
23                      self.__db_obj.lock_it()
24                      srbserver_id = self.__db_obj.insert_srbserver(d_xml[machine],self.
                            __srb_port)
25                      srbserverid_set[machine] = srbserver_id
26                      self.__db_obj.rlock()
27                  else:
28                      if self.__i_verbose:
29                          print "no data there for machine:",machine
30                      if last_insert.has_key(machine):
31                          if last_insert[machine] == d_xml[machine]["REPORTED"]:
32                               print "No fresh data from Ganglia for %s, maybe server lost
                                    " % machine
33                      break
34
35              self.__db_obj.lock_it()
36              is_id = self.__db_obj.insert_iteration_srbserver(self.__iteration_id,
                    srbserverid_set[machine])
37
38              if last_insert.has_key(machine):
39                  if last_insert[machine] == d_xml[machine]["REPORTED"]:
40                      print "No fresh data from Ganglia for %s, maybe server lost" %
                            machine
41              last_insert[machine] = d_xml[machine]["REPORTED"]
42              host_count +=1
43              self.__db_obj.insert_measurement(d_xml[machine], is_id,d_xml['localhost'])
44              self.__db_obj.rlock()
45
46          if host_count==len(d_xml.keys())-1:
47              self.__db_obj.set_all_measurements_there("set")
48              if (self.__measurement_app != "none" or self.__measurment_app != "") and
                    application_started == 0:
49                  self.__run_app_thread = Crunapp(self.__measurement_app, self.__i_verbose
                        )
50                  self.__run_app_thread.start()
51                  if self.__i_verbose:
52                      print "Test application started!!"
53                  application_started = 1
54
55          counter += 1
56
```

```
57          if counter > int(self.__quantity):
58              if self.__i_verbose:
59                  print "stop the measurement"
60              self.stop()
61              time.sleep(2)
62              break
63          sleep_counter =0
64          while sleep_counter != self.__poll_time:
65              time.sleep(1)
66              if self._stopevent.isSet():
67                  break
68              sleep_counter +=1
69          if self._stopevent.isSet():
70              break
```

6.5 Multiple Measurement Diagram

The last section of the “Implementation” chapter shows how a list will be created for the Graph module to draw a history diagram. As already mentioned before, there are two different views and this section will confine itself on the “Multiple view”. This view draws diagrams for more than one measurement cycle, more than one SRB server, or both. The "__create_view_multiple" function in listing 6.11 on page 79 is part of the Graph class and is stored in the myplot module. The function receives a list parameter from the database with all measurements of the measurement cycle, which have been done so far. The task is to convert the given list into a prepared list for the external "Graphs" module. The list has to store the values of the monitored applications and machines. That means, the list combines the measurement values of the application table and the measurement table are for one measurement cycle.

The Graphs module needs at the end a list, which has the following structure:

```
list = [(30, 2), (31, 4), (32, 4),
(33, 2), (34, 2), (35, 2), (36, 2),
(37, 2), (38, 2), (39, 3), (40, 3)]
```

Therefore, a two element “tupel” has to be stored in an one dimensional list. The first element represents the x-value and the second the y-value of a point in the graph. The given list example looks like figure 6.1.

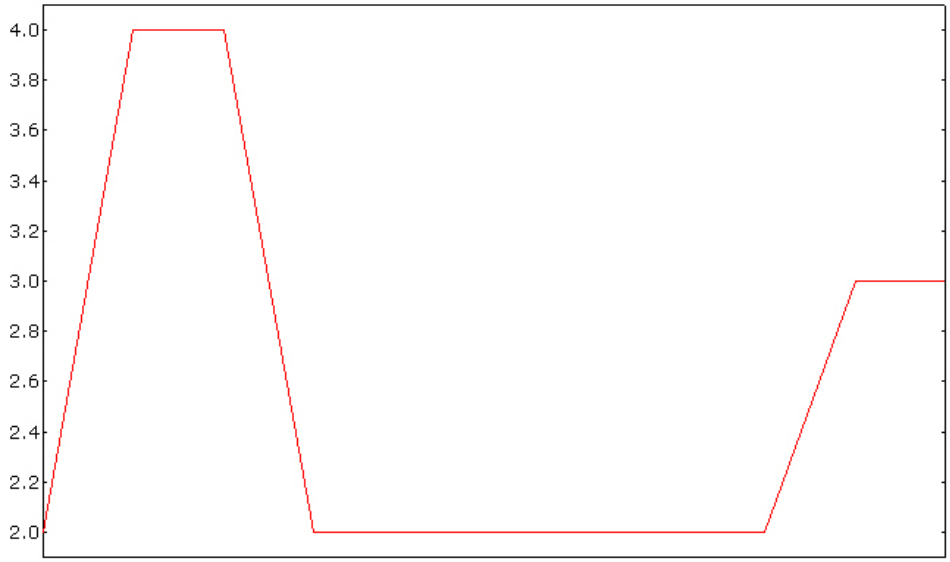


fig. 6.1: GUI Example

The function __create_view_multiple converts the measurements and structures them in several lists. The function starts in line 13 with a “for” loop, which has length of the given measurement cycles. The line 15 till 23 get main values from the host, srbserver and iteration table for one particular measurement cycle and SRB server combination in the database. There are different metrics, which all have to be sorted in different lists. Hence, all the list will be stored in a dictionary, where all values are sorted by the metric name and the is_id of the iteration_srbserver table. The self.__itersrbserv_comb dictionary stores all the list and the first key will be the metric name and the second the is_id. The is_id points at a couple of measurements, which finally represent the graph. The ID is used, because it allows to recover the SRB server and measurement cycle from one particular measurement. In line 26 the start time of the measurement cycle will be stored, because all other times will be subtracted from this time to create the x-value for the graph. The line 33 creates the different dictionaries for every metric of the measurement table.

```
self.__itersrbserv_comb[server_cpu_system][10]
= [(0, 2), (15, 4), (30, 4)]
self.__itersrbserv_comb[server_cpu_system][14]
= [(0, 2), (15, 4), (30, 4), (45, 1)]
```

The example shows a list for the metric server_cpu_system and two different is_id keys. Both graphs will be shown in one diagram and the first graph is three and the second four points long. That is the way all measurements are stored in one dictionary and it allows to change the database to add more metrics without any need of modifying of the GUI modules.

fig. 6.2: Listbox

The for loop in line 35 stores all metrics of the measurement table in the dictionary. Afterwards in line 44, the application table will be queried and all application metrics will be stored in the dictionary from line 50 till 60. The key, which will be used to store the application metrics, is built out of the application name and the metric. For example the bash application and the open file descriptor metric is stored under the "bash_fd" key. The last part of the function is to save the first key of the self.__itersrbserv_comb dictionary in the list box of the diagram dialog. So the picture 6.2 shows the keys, which are used to browse through the different diagrams. The picture 5.9 on page 55 of the Design chapter will help to understand the diagram dialog, because it shows a multiple view with graphs of different machines and different measurement cycles in one diagram.

Listing 6.11: Multiple view

```
1     def __create_view_multiple(self,list):
2         """ Create the view for more than one test (reused test)"""
3
4         # set the view to multiple view
5         self.__view = "multiple"
6         self.__itersrbserv_comb = {}
7         count = 0
8         self.start_time = {}
9         self.multilegend = {}
10
11        # get some information for the legend, so the graphs can be relocated
12        # furthermore store the data in a dictionary for the graphs
13        for list_elem in list:
14
15            self.__db_object.lock_it()
16            self.__db_object.db_cursor.execute("""SELECT I.time, H.hostname, S.srb_port,
                  I_S.is_id FROM
17            iteration_srbserver AS I_S, srbserver AS S, iteration AS I, host AS H, test
                  AS T
```

```
18              where I_S.is_id = %s AND I_S.srbserver_id = S.srbserver_id
19              AND S.host_id = H.host_id AND I.iteration_id = I_S.iteration_id
20              AND I.test_id = T.test_id  """ % list_elem["M.is_id"])#, T.name,
21
22              test_time = self.__db_object.db_cursor.fetchone()
23              self.__db_object.rlock()
24              is_id = str(test_time[3])
25              if not self.start_time.has_key(is_id):
26                  self.start_time[is_id] = list_elem["M.time"]
27                  self.multilegend[is_id] = (test_time[0],test_time[1],test_time[2])
28              if count == 0:
29                  count = 1
30                  keys = list_elem.keys()
31                  s = re.compile("(.*)(time|_id$)")
32                  for x in keys:
33                      if not s.match(x):
34                          self.__itersrbserv_comb[x[2:]] = {}
35              for x in keys:
36                  if not s.match(x):
37                      if not self.__itersrbserv_comb[x[2:]].has_key(is_id):
38                          self.__itersrbserv_comb[x[2:]][is_id] = []
39                      self.__itersrbserv_comb[x[2:]][is_id].append((time.mktime(time.
                            strptime(list_elem['M.time'],'%Y-%m-%d %H:%M:%S'))\
40                      -time.mktime(time.strptime(self.start_time[is_id],'%Y-%m-%d %H:%M:%S
                            '))\
41                      ,float(list_elem[x])))
42
43              self.__db_object.lock_it()
44              self.__db_object.db_cursor.execute("SELECT A.* FROM application AS A,
                    measurement AS M WHERE A.measurement_id = %s \
45                  and M.measurement_id = A.measurement_id" %list_elem["M.measurement_id"])
46              applications = self.__db_object.db_cursor.fetchall()
47              self.__db_object.rlock()
48
49              # store the values for the particular time in is_id
50              for x in applications:
51                  for key, item in x.items():
52                      if key != "A.name" and key != "A.command" and key != "A.
                            application_id" and key != "A.measurement_id":
53                          if not self.__itersrbserv_comb.has_key(x['A.name']+"_"+key[2:]):
54                              self.__itersrbserv_comb[x['A.name']+"_"+key[2:]] = {}
55                          if not self.__itersrbserv_comb[x['A.name']+"_"+key[2:]].has_key(
                                is_id):
56                              self.__itersrbserv_comb[x['A.name']+"_"+key[2:]][is_id] = []
57                          self.__itersrbserv_comb[x['A.name']+"_"+key[2:]][is_id].append((
                                time.mktime(\
58                          time.strptime(list_elem['M.time'],'%Y-%m-%d %H:%M:%S'))\
59                          -time.mktime(time.strptime(self.start_time[is_id],'%Y-%m-%d %H:%
                                M:%S'))\
60                          ,float(item)))
61
62          count = 0
```

```
63
64          # setup the listbox with possible graphs
65          if self.__listb.size() == 0:
66              for x in self.__itersrbserv_comb.keys():
67                  self.__key_list.append(x)
68              self.__key_list.sort()
69              for x in self.__key_list:
70                  self.__listb.insert("end",x)
71              active = x
72
73          # store the current graph in self.__active
74          if self.__active == None:
75             # self.lists = self.__itersrbserv_comb[active]
76              self.__active = active
```

Chapter 7

Tests

The measurement system has been tested with one and three machines. Unfortunately, a testing SRB system could not be provided until the end of the project. Furthermore, the development and the finished prototype of the project needed more time than expected. At least the measurement system was tested on an SRB system running on one machine with a monitoring client and a monitoring server. The following test scenarios have only the purpose to show that the required functionality has been achieved.

7.1 Test with one Server

The test environment is a Samsung P35 notebook with a 1.6 Giga Hertz (GHz) Intel® centrino processor. A Linux Suse distribution version 9.3 is installed on the machine and will be used to perform the test. All necessary tools are installed in the way as it has been described in the "Installation and Configuration" section A.2 on page XIII. The SRB server, the monitoring client, and the monitoring server are running on this machine.

7.1.1 Configuration

At first, the SRB server needs to be started. Thus, the SRB server will be executed with the command "`perl install.pl start`". Afterwards the monitoring server will be started without any parameter that the network port will be set by default to 5000. The last step is to start the GUI client and to configure the measurement in the configuration file or directly in the client. The most important setting in the configuration file is the IP address

and the network port of the monitoring server. The configuration file used for the test has been adjusted as follows:

Listing 7.1: Test Configuration for 1 Machine

```
1  [test]
2  application = python2.4; postmaster
3  name = lasttest
4  desc = none
5
6  [project]
7  name = Testproject
8  desc = none
9
10 [tester]
11 surname = Koebernick
12 email = c.koebernick@rl.ac.uk
13 forename = Carsten
14
15 [server]
16 host_1 = 192.168.12.14
17 port_1 = 5000
18 quantity = 1
19
20 [measurement]
21 srb = 1
22 application = python2.4 /home/ck/programs/SputTests/src/sputData.py
23 poll_time = 15
24 quantity = 28
```

It is important to set the measurement → srb value to 1 for measuring the SRB server. For more information about the configuration file, please have look back in the table 5.1 on page 43.

By pressing the "Start" button the test begins in the monitoring client. For testing the measurement of the SRB application, a given script from the RAL has been used. The script creates load on the SRB application for approximately four minutes. Therefore, the test will measure for seven minutes to see how the applications behave during and after the measurement has finished. The monitoring client measures will measure the the applications every 15 seconds, which means the maximum number of measurements must be 28. Three applications will be measured, the SRB application, the "Postgres" database, which is part of the SRB application, and the test application from the RAL.

7.1.2 Results

The picture 7.1 shows that the SRB server does not create the main load. The biggest load will be created by the "postmaster" process, stands for the Postgres database. The "Python2.4" graph is the test application and creates also less load than Postgres.

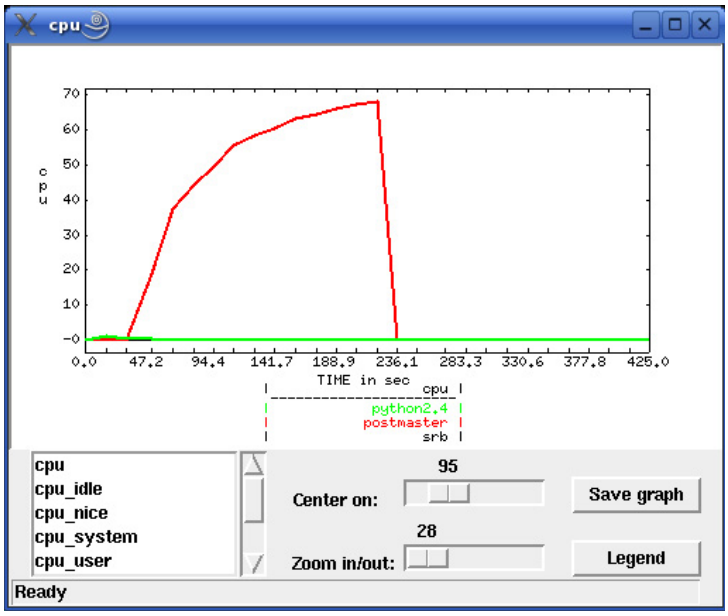


fig. 7.1: Test Single View

The maximum values of the measurement cycle are listed in table 7.1.

Table 7.1: Maximum values of the applications

Application	CPU load	Memory	File descriptors
Test application	1.3	0.2	9
SRB	0.2	1.0	32
Postgres	68.1	2.7	306

Unfortunately the complete load is on one machine, which means the measurement system will be influenced by the load of the SRB server. Hence, the 15 seconds between two measurements cannot be maintained. Mostly the time interval between two measurements is 16 seconds, but up to 19 seconds are possible.

The table 7.2 shows that the measurement has used a maximum of 75% CPU system capacity. Furthermore it is interesting that a system, which runs a Standard Suse 9.3 distribution with a few programmes and an X server[1], runs over 100 processes on a Linux operating system. The other results just show that the application works properly.

Table 7.2: Maximum values of the machine

Metric	**Value**
cpu_system	75.0
cpu_nice	0.0
cpu_user	22.0
cpu_idle	83.1
proc_total	131

Furthermore, a measurement cycle with 200 measurements has been tested and successfully finished. So, if someone needs a measurement system running on a single machine, which collects the CPU load and other metrics from the SRB server and other applications and provides history and real time graphs, this measurement system can be used.

[1] X window system provides an API between hardware and a desktop systme to provide a GUI on Unix based systems[23]

7.2 Test with three Servers

As mentioned before, this test runs no SRB server application and measures only standard applications in a network with three computers. The monitored application is bash, because this application ran on all machines. All computers are in a C-class network, which is in the range of 192.168.1.0 and 192.168.1.254. The used computers are in table 7.3.

Table 7.3: Machine Specification

Component	IBM T42	IBM T22	Samsung P35
CPU	Pentium M 1.6 GHz	Pentium 3 900 MHz	Pentium M 1.6 GHz
Memory	512 MB	256 MB	1280 MB
Chipset	Intel 855PM	Intel 440BX	Intel 855PM
IP address	192.168.1.102	192.168.1.103	192.168.1.106
Host name	theodore.anderl	localhost.localdomain	linux.site
Distribution	Suse 9.3	Debian sarge	Suse 9.3
Task	monitoring server	monitoring server	monitoring server & client

7.2.1 Configuration

The server machines, which are all three computers, need to have Ganglia and a standard Python version running the monitoring server. The Samsung computer runs again the monitoring client, which connects to its own monitoring server and to the other ones.

It is very important, before testing the measurement system in a network, to look for the distribution firewall of Linux. Sometimes a Firewall is running, which blocks the Ganglia UDP transmission. For example, the “SuseFirewall” of Suse 9.3 has blocked Ganglia, but not the TCP connection between the monitoring client and the monitoring server. Therefore, the user might wonder why the measurement system does not receive any data, whereas it is connected to the servers. So, either stopping, or configuring the firewall is necessary that the system is able to work properly.

The configuration file has to be adjusted again. The application, which should be monitored, must be changed to “bash” to monitor both applications. The srb flag can be set

to zero, because no SRB server is running and the three servers must be configured with port and IP address.

```
...

[measurement]
srb = 1
application = none
...

[test]
application = bash; gkrellm
srb = 0
...

[server]
host_1 = 192.168.1.106
port_1 = 5000
host_2 = 192.168.1.102
port_2 = 5000
host_3 = 192.168.1.103
port_3 = 5000
...
```

7.2.2 Results

The Maximum values of all three machines are shown in table 7.4. If the monitored applications occur mistakes, all measurements will be 0. All three servers idle most time and the Bash runs on the computers with less CPU usage. The test results show that it might be useful to know how much processes are running of the respective application. For example the "T42" has 84 open file descriptors, which means for the Bash that many processes are running of the application.

Table 7.4: Maximum Values of 3 Machines

Metric	**IBM T42**	**IBM T22**	**Samsung P35**
bash_cpu	2.1	1.8	0.7
bash_mem	4.1	2.8	0.5
bash_fd	84	21	23
server_cpu_idle	90.0	77.0	80.0
server_cpu_nice	0.0	0.0	0.0

Continued on next page

Metric	IBM T42	IBM T22	Samsung P35
server_cpu_user	4	40	28
server_cpu_system	3.6	6.0	6.5
server_proc_total	113	87	139

The picture 7.2 shows the associated graph for the “server_cpu_system”. The user should not use more than 4 graphes in one diagram, because the diagram would become to complex.

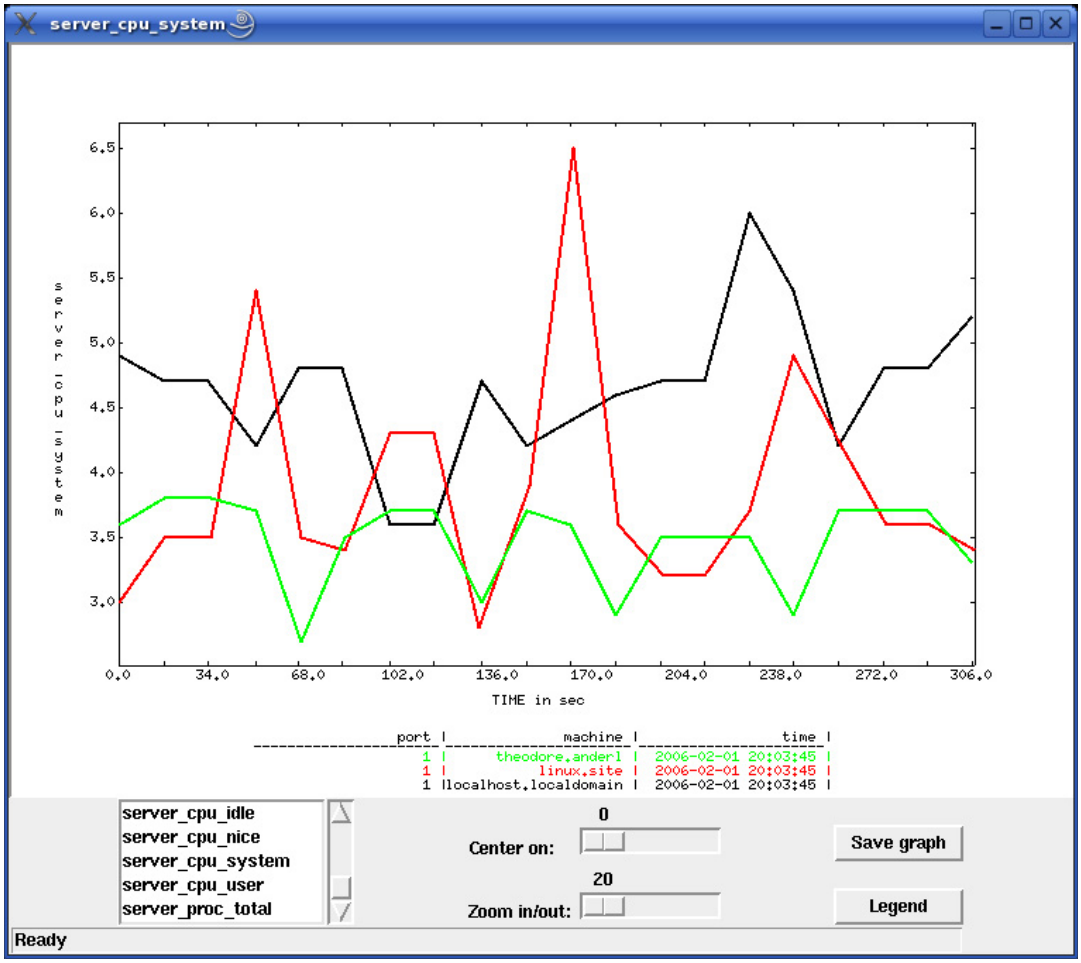


fig. 7.2: Test Multiple View

Chapter 8

Conclusion

The project has finished with a prototype measurement system that allows to monitor SRB and standard applications. The measurement system is based on a client server approach, which is one possibility to meet the task of the project. Many features were developed during the implementation of the project. For example, the dynamic graph was not part of the project, but helps getting a general survey over the current measurements. Furthermore, two client applications have been developed to work with and without a Graphical User Interface. So, the measurement system can be used independent from a Linux Desktop environment and the TK programming language. Both clients can be used for long-run measurements to monitor machines for hours and days. The amount of measurements can be adapted over the time interval between two measurements and the maximum number of measurements. So after the system has been configured it can run without any interaction of the user. Imaginable uses are that the console client does the measurement and the user checks the current measurement cycle with the GUI client. Besides the use of the applications the code has been divided into modules, that for example the socket connection can be used for other projects as well. Furthermore two dialogs were developed to separate diagrams and main GUI. So, the system can open endless diagram dialogs to compare every different measurement cycle and machine from every project or test with each other.

The development has also shown that other approaches are also qualified to solve the project. The client server approach needs to start and stop a monitoring server on the server machines. Instead of using a monitoring server, a daemon could be started, which would be configured with a configuration file. That means, the measurement system could

abandon on the additional socket connection to configure the server machines. But, there are always many ways leading to Rome.

Unfortunately, not all targets have been achieved. The adding of new metrics needs little code and database changes, which is not efficient. However, the main target was at first to finish all features. So, for the improvements of the system was no time at the end of the project. The now following "Future Prospect" shows a new database approach, which would solve the problem.

Chapter 9

Future prospect

The following chapters gives some suggestions for an improvement of the measurement system. Most important changes are necessary in the database model and from it following code changes. Furthermore, some features will be given, which could improve the handling of the measurement system.

Debugging and Testing As already mentioned, the measurement system needs more tests and debugging sessions. Especially an SRB system is needed, where some measurements run with more than monitoring server. The programme needs also more security queries to develop a more reliable system.

Look & Feel The usability of the main dialog can be improved in many parts. For example, the opening and closing of the list boxes in the main frame might be confusing for a new user. Furthermore, a simple view, which allows a closer look on the tables, has already been started, but not finished until the end of the project. It would be nice, if the user could edit the entries of the test, project, tester, and host table, so that it is possible to change for instance the description of a particular project.

User help inside of the GUI application If the measurement system should get more acknowledgments, it would be nice to have a Help screen in the GUI, which explains the main features directly in the application. A help frame would give the user a little introduction into the measurement system and the user will get a simpler access to the system.

Smaller functions in the gui2_ui module The gui2_ui module, which contains the classes for the main dialog, needs smaller functions. Especially the creation of the list

boxes within the graph_choice class has to long functions, which makes it hard to extent the dialog with more functionality. Therefore, this module needs a review.

Add y-axis zooming to the Graph dialog Now, the measurement system allows zooming into the history diagrams, but only within the x-axis. The y-axis zooming would be good to scale into the diagram and take accurate measurements. If a measurement cycle consists of values with big value differences, only inaccurate metering is possible.

Change the timing behaviour The timing between two measurements is not accurate in comparison to the value in the configuration file. The load of other applications can effect the time. Thus, it is better to create a time manager, which uses clock time instead of using the sleep function. Furthermore, the latency that occurs between two measurements effects every further measurement. For instance, if the time interval between two measurements is set to 15 seconds and the first measurement has been done at 14:30:15, the measurement system would wait until 14:30:30 before the next measurement will be made.

Migrate the application to newer versions of Python, SQLite and Pysqlite Some functions missed during the development, which would allow a foolproof implementation of the measurement system. For instance, the socket module of Python 2.3 supplements with a function to set a connection timeout. If a particular IP address is available, but the port is not open, the connection waits until the operating system returns that the connection has not established. However, the connection timeout can be set to a particular time interval with Python 2.3. For example, if the server and the client have no connection within that time interval, the function would stop the connection establishment after the time ran off.

If a new version of Python would be used, the SQLite and Pysqlite versions should be upgraded as well. SQLite version 3 allows writing and reading at the same time. There are still some bugs with multithreaded environments, but the "Changelog" says that SQLite3 at least deals better with threading [24]. If it works better, it should be considered to abandon on the exclusive locks for every database command. This might improve the speed of the application, because sometimes the main dialog of the measurements needs many database queries successively and this means many locks and releases. The old SQLite 2 databases can be kept, because database files can be converted to the new SQLite3 format and back:

```
sqlite3 olddb.db .dump | sqlite newdb.db (sqlite3->sqlite2)
sqlite olddb.db .dump | sqlite3 newdb.db (sqlite2->sqlite3)
```

A good side effect is that the file size would be decreased between 25% and 35%.

The used Pysqlite 1.0 for SQLite2 is outdated. The new version Pysqlite 2.1 should be used for the SQLite3 datbase. Other interfaces between Python and SQLite become available with the new versions, such as "apsw"[25], which is smaller than Pysqlite.

Database revision The database needs a little more normalisation, because there are still some redundancies in the database. For example, the measurement table stores the

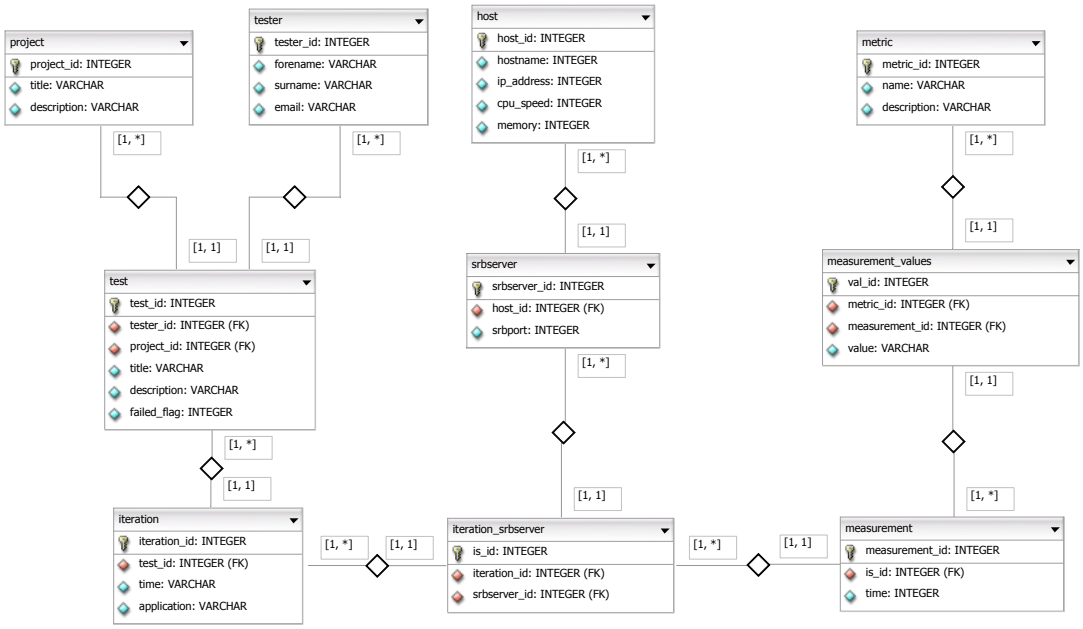


fig. 9.1: New Database Approach

measurements for every SRB server and a few measurements for the client. If more than server will be measured in one measurement cycle, the data of the client will be stored in every measurement row of the measurement table for every SRB server. This means the number of servers determines the number of redundancies. So a possible solution is to create an own measurement row for a client and abandon the client columns from the measurement table. The data would be independent from the SRB server measurements.

Furthermore, complete new tables are conceivable. For example a metric table, which stores all types of metrics so far used and will be advanced if new types of metrics have been used. The value table would store the values of the measurement and a foreign key would link the value to the type of metric. The figure 9.1 on the page before shows the new approach. The modification of the database would need many changes in the code, but the measurement system would be very easy to extent with new type of metrics.

Plugin Module for new metrics Another improvement could be a new Python module, which parses two folders containing Bash scripts. The first folder consists of scripts to monitor applications and the other one to monitor the machine. This enables an easy extension of new metrics.

Abbreviations

API	Application Programming Interface
bash	Bourne Again Shell
gmetad	Ganglia meta daemon
gmond	Ganglia monitor daemon
GPL	General Public License
Gtk	Gimp Toolkit
GUI	Graphical User Interface
MFC	Microsoft Foundation Classes
SDSC	San Diego Supercomputer Center
SQL	Structured Query Language
SRB	Storage Resource Broker
TCL	Tool Command Language
TCP	Transmission Control Protocol
TK	Toolkit
UDP	User Datagram Protocol
XDR	External Data Representation
XML	Extensilbe Markup Language

Bibliography

[1] *Perl Snapshot - Kurzer Prozess*. http://www.linux-magazin.de/Artikel/ausgabe/1999/02/Proc/proc.html.

[2] *Ganglia webpage*. http://www.ganglia.info.

[3] *Nagios webpage*. http://www.nagios.org.

[4] *Genral Public License*. http://www.gnu.org/licenses/gpl.html.

[5] *Wikimedia Ganglia Server*. http://ganglia.wikimedia.org.

[6] *RRDtool*. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/.

[7] *NagiosExchange Portal*. http://www.nagiosExchange.org.

[8] *PerfParse - Addon for Nagios*. http://perfparse.sourceforge.net/.

[9] *mySRB*. http://www.sdsc.edu/srb/mySRB/mySRB.html.

[10] *inQ*. http://www.npaci.edu/dice/srb/inQ/inQ.html.

[11] *Grid Security Infrastructure*. http://security.sdsc.edu/help/pki/gsi.shtml.

[12] *SDSC, SRB - Scommands*. http://www.sdsc.edu/srb/srbcommands.html.

[13] *Understanding file permissions on Unix: a brief tutorial*. http://www.dartmouth.edu/~rc/help/faq/permissions.html.

[14] *TkInter*. http://wiki.python.org/moin/TkInter.

[15] *SQL development tutorial*. http://www.cachemonitor.de/sql.html.

[16] *SQLite documentation*, 2005. http://www.sqlite.org/docs.html.

[17] *MySQL for Python*. http://dustman.net/andy/python/python-and-mysql.

[18] *Pysqlite homepage*. http://initd.org/tracker/pysqlite.

[19] *XML Meta languages*. http://www.brics.dk/ixwt/IXWT_C04c.pdf.

[20] *SQL - FAQ*. http://sqlite.org/faq.html#q8.

[21] *UNIX ON-LINE Man Pages*. http://unixhelp.ed.ac.uk/CGI/man-cgi.

[22] Mark Lutz & David Ascher. *Learning Python*. O'Reilly, 2. edition, 2004.

[23] *X window system*. http://www.xfree86.org/.

[24] *SQLite 3*. http://www.sqlite.org/version3.html.

[25] *APSW - Another Python SQLite Wrapper*. http://www.rogerbinns.com/apsw.html.

[26] *Bash Reference Manual*. http://www.gnu.org/software/bash/manual/bashref.html.

[27] *Ethereal webpage*. http://www.ethereal.com.

[28] *The Ganglia Distributed Monitoring system*. http://www.clusterworld.com/CWCE2004/Matt_Massie_presentation.pdf.

[29] *fraendz - Install Sqlite*. http://www.stud.uni-goettingen.de/~s291325/cgi-bin/run-cvstrac.cgi/wiki?p=InstallSqlite.

[30] *KDE.org*. http://www.kde.de/index.php.

[31] *Python homepage*. http://www.python.org.

[32] Helmut Herold. *Linux/Unix-Kurzreferenz (short reference)*. Addison Wesley, 2. edition, 2003.

[33] *SQLite download*. http://www.sqlite.org/download.html.

[34] SDSC, http://www.sdsc.edu/srb/CurrentSRB/SRB.htm. *SRB - Storage Resource Broker*, current edition. San Diego Supercomputer Center.

A Handbook

A.1 Files

The following subsections list all necessary files to use the measurement system.

A.1.1 Monitoring server

Table A.1: Files of the monitoring server

File	Language	Task
python_server.py	Python	executable
ganglia.py	Python	connection to gmond
socket_connection.py	Python	provides a socket connection
average.sh	Bash	measures the CPU efficiency of the SRB server
average_mem.sh	Bash	measures the memory efficiency of the SRB server
num_fd.sh	Bash	measures the open file descriptors of the SRB server

A.1.2 Monitoring client

Table A.2: Files of the monitoring client

File	Language	Task
gui2.py	Python	executable for the GUI client

Continued on next page

File	Language	Task
console.py	Python	executable for the console client
python_client.py	Python	main functions for the measurement cycle
gui2_ui.py	Python	main dialog of the GUI
myplot.py	Python	second dialog for the diagrams
tooltip.py	Python	tooltip class for the diagram dialog
Graphs.py	Python	provides the diagrams for the myplot module
utils.py	Python	small scripts for the Graphs module
config.ini	Text	configuration file
ganglia.py	Python	connection to gmond
socket_connection.py	Python	provides a socket connection
srb.db	SQLite	database file; can be created with the executables

A.2 Installation and Configuration

The installation of the necessary tools was one of the first investigations of the project. All these applications have to be installed in user mode. This means no precompiled binary packages, which are common for Linux distributions such Suse, Red Hat, or Debian, can be used for the installation. The source packages of Ganglia, SQLite, Python, and Pysqlite have been downloaded from the respective homepages and the following versions have been used:

Table A.3: Application Versions

Application	Version	Reference
Python	2.2.3	[31]
Ganglia	3.3.1	[2]
SQLite	2.8.16	[33]
Pysqlite	1.0	[18]

The monitoring server needs only a Python and Ganglia installation, but the client needs all four.

The standard progress for installing applications in Linux is to `configure` the sources, build the sources with `make` and install them with `make install`. Many applications can be installed in user mode by adding a single parameter to the `configure` command. The parameter `--with-prefix="/<directory>/"` sets a new directory for the installation path and allows to install the sources into a folder with user rights. The following progress works for most applications:

```
./configure --prefix=/home/<username>/<program folder>
./make
./make install
```

Not every application provides a configure script. Some source packages use installation files written in particular script languages. Therefore, the following subsections show the installation of the used tools in more detail.

A.2.1 Ganglia

The installation of Ganglia is the same as it has been explained before. Just use the following commands to install it:

```
./configure --prefix=/home/%username%/%program folder%
make
make install
```

Afterwards it is necessary to create “gmond.conf” to test the correct installation:

```
%ganglia-folder%/sbin/gmond -t > gmond.conf
```

After the creation one need to change the parameter `setuid=yes` to `setuid=no` and delete the line `user=nobody` in the file. Furthermore, Ganglia should not use a multicast channel, because measurement system uses direct connections to minimise the traffic. Otherwise all measurement data would be send to every monitoring server and client. By substituting the `mcast` option with the `host=<hostname or IP>` option, Ganglia will be ready for the first test. The host option can be set to the localhost address: `127.0.0.1`. The following command starts the gmond with configured “gmond.conf”:

```
./gmond -c gmond.conf
```

An additional parameter `-d %debug-level%` starts the gmond as a standard application and not as daemon. Afterwards, the gmond shows verbose messages, which can be analysed, if problems occur. To gather the output of Ganglia, a telnet query can be used to test the functionality.

```
telnet 8649 localhost
```

The command queries the localhost address on port 8649 for any messages and if the gmond runs properly it will return an XML string with the standard metrics that are set in the "gmond.conf".

A.2.2 Python

Python is also very easy to install, because it works with the standard installation. The only parameter, which should be given to the configure script is `--with-Tkinter`. This enables the installation of the Tkinter library, which is by default part of the Python library. It is necessary if the user wants to use the GUI monitoring client. This installation manual supposes that TK has been installed, before Python. It is normally the case on standard Linux distributions. The Python installation needs only the TK libraries.

A.2.3 SQLite and Pysqlite

SQLite needs no further explanation, because it works with the standard installation. It is wise to use the option `--enable-threadsafe`, because the client application uses threads to parallelise tasks.

Pysqlite uses an Python script for installation, which needs some adjustments:

```
# edit setup.py
```

```
    * include_dirs['/%sqlite-folder%/include']
    * library_dirs['/%sqlite-folder%/lib']
    * runtime_library_dirs=[]
```

```
    * runtime_library_dirs=library_dirs

# %python_path%/bin/python2.2 setup.py build
# %python_path%/bin/python2.2 setup.py install
```

At first the user has to edit the “setup.py” script and change the SQLite directory paths with the one on his system. Afterwards the “setup.py” has to be started with option build and then with the option install. It is very important to use the Python version that has been installed by the user. There might be a root version already on the system. Therefore, it is recommendable to use the complete Python path for the installation of Pysqlite. To test the installation of Pysqlite the tool “pydoc” is very useful. Pydoc is in the binary folder of the Python directory and can be started with the command line `pydoc -p <port>`. All the installed packages, modules, and classes can be accessed with a browser and the target “localhost:<port>”. If Pysqlite has been installed correctly, there should be an “sqlite” package. [29]

A.2.4 Measurement system

Before starting the monitoring client and server, both applications need to know where Ganglia is located. Therefore, a small change in the python_server and python_client module is necessary. Both modules begin with the import of particular library modules and afterwards the assignment of the global variable `ganglia_directory` follows. It is necessary to change the value to the main Ganglia folder on the machine. For example:

```
ganglia_directory = "home/user1/programs/ganglia/"
```

Furthermore the python_client module needs the name of the gmond configuration file, which must be located in the `sbin` directory within the main ganglia folder.

```
gmond_conf_file = "gmond_test.conf"
```

Those are all necessary changes on the modules and the measurements can begin.

A.3 Monitoring Server

A.4 Console Client

Table A.4: Console Client Parameter

Parameter	Exercise
primary-keys	
-p, –project-id <id>	passing a project id
-t, –test-id <id>	passing a test id
-i, –iteration-id <id>	passing an iteration id
-s, –srbserver-id <id>	pass a srbserver id
tables	
–projectlist	draw a project-table
–testlist	draw a test-table
–iterationlist	draw an iteration table
–srbserverlist	draw a srbserver table (including host parameter)
–measurementlist	draw a measurement table
other parameter	
-s, –startday <YYYY-MM-DD>	used in combination with endday to get measurements between two dates
-e, –endday <YYYY-MM-DD>	
-d, –database <file>	pass a database (standard:srb.db)
-c, –config-file <file>	pass a config file (standard: config.ini)
-f, –config-dump <file>	store an config example in a new config file
-g, –gauge	start a measurement
-h, –h, –help	draw the usage of the client
-v, –verbose	draw more messages during the process of the client application

A.5 GUI Client

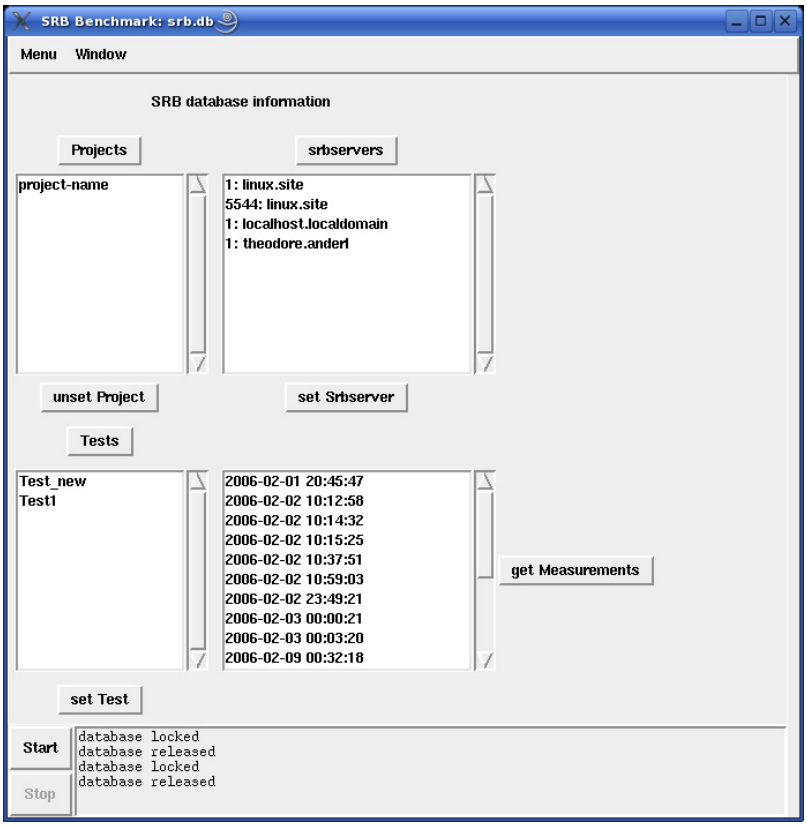

SRB Benchmark: srb.db
Menu
Window
SRB database information
Projects
srbservers
project-name
1: linux.site
5544: linux.site
1: localhost.localdomain
1: theodore.anderl
unset Project
set Srbserver
Tests
Test_new
Test1
2006-02-01 20:45:47
2006-02-02 10:12:58
2006-02-02 10:14:32
2006-02-02 10:15:25
2006-02-02 10:37:51
2006-02-02 10:59:03
2006-02-02 23:49:21
2006-02-03 00:00:21
2006-02-03 00:03:20
2006-02-09 00:32:18
get Measurements
set Test
Start
Stop
database locked
database released
database locked
database released

fig. A.1: Main

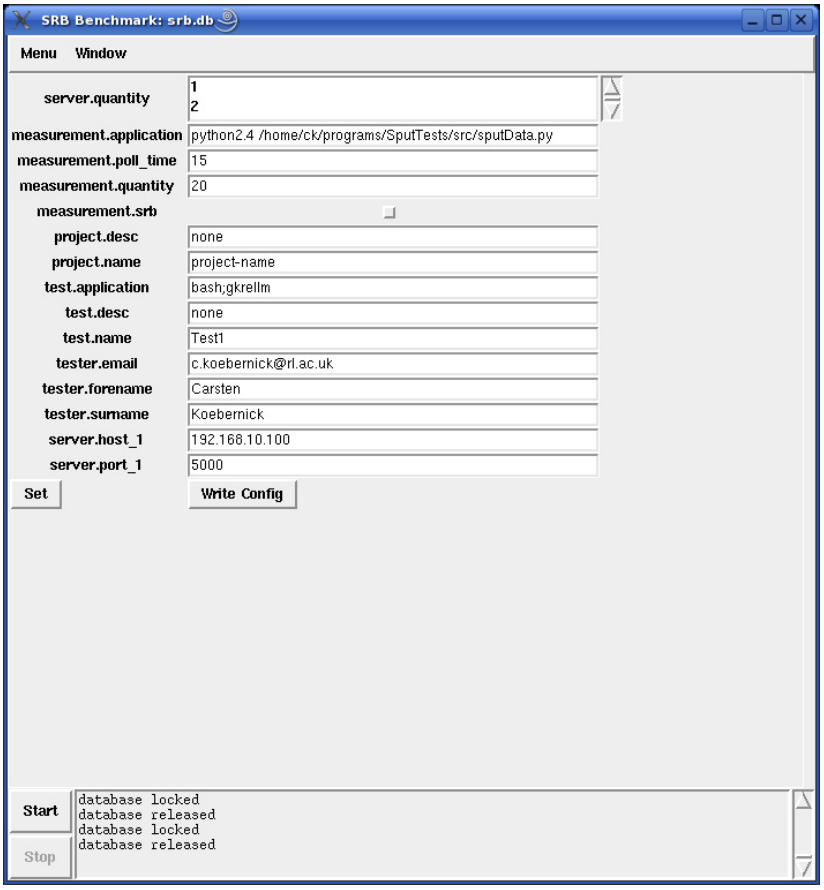


fig. A.2: Configuration

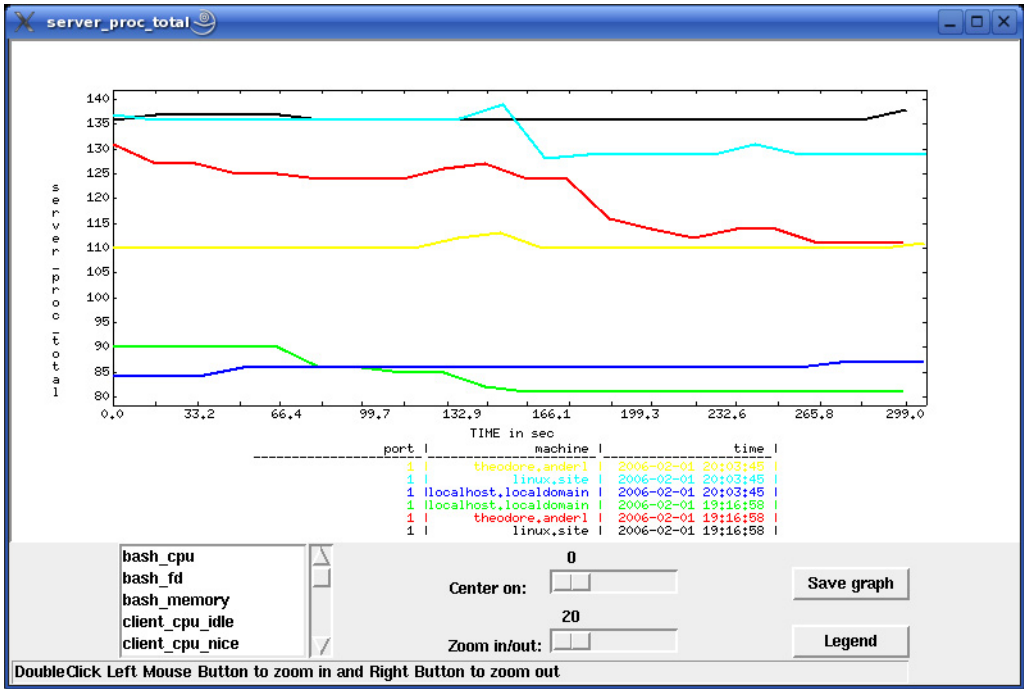


fig. A.3: Diagram Dialog

B Python Scripts

B.1 python_server.py

```
#!/usr/bin/env python



""" Performance test program (Server) to catch
information about applications running on the SRB """

__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "07.09.2005"
__version__ = "0.1"
__revision__ = "1.0"
################ never use more than one server on one machine ######################

""" the module commands is necessary to get the output from the telnet into a variable """
import commands

""" import the server class from the file socket_connection.py """
from socket_connection import CsocketServer
""" sys is used to stop the application with sys.exit() """
import sys

import getopt

import threading

import re

from ganglia import Cinitialization
from time import sleep

i_portnumber = 5000
### change here to setup another ganglia folder
ganglia_directory = "~/programs/ganglia/"

class Cgetoptions:
    """ get options and parameter from the command line"""
    def __init__(self):
        """ constructor for Cgetoptions: initialise the parameter"""
        try:
            self.__opts, self.__args = getopt.getopt(sys.argv[1:], 'p:hv', ["help", "port=", "verbose"])
        except getopt.error:
            print "argument mistakes"
            self.__usage()
            sys.exit(-1)

    def __usage(self):
        """ Help output for new users """
        print "Usage"
    def start(self):
        """ start the parameter verification """
```

```
        d_commands = {}
        d_commands['verbose'] = 0
        for s_option, s_argument in self.__opts:
            if s_option in ("-p", "--port"):
                d_commands['port'] = s_argument
                reg_obj = re.match("([^0-9])|((\D)*(\d)+(\D)+)", s_argument)
                # just numbers are allowed for the port

                if reg_obj != None:
                    print "port needs to be a number"
                    sys.exit(0)
                i_argument = int(s_argument)
                if (i_argument < 1025) or (i_argument > 65535):
                    print "the port needs to be between 1025 and 65535"
                    sys.exit(0)
            #print verbose messages
            elif s_option in ("-v","--verbose"):
                d_commands['verbose'] = 1
            #print help (self.usage())
            elif s_option in ("-h", "--help"):
                print self.__usage()
                sys.exit(0)
            #parameter were not correct
            else:
                print "bad parameter"
                print self.__usage()
                sys.exit(0)
        if d_commands.has_key('port') == False:
            d_commands['port'] = 5000
            print "set port to 5000"
        return d_commands

class RefreshGanglia(threading.Thread):
    """ Thread to refresh the information of Ganglia """

    def __init__(self, init_obj, srb_port, application_name, i_verbose = 0):
        """ constructor to initialise the thread and create the stop event """
        threading.Thread.__init__(self, name = "Refrehsgmond")
        self.i_verbose = i_verbose
        self.init_obj = init_obj
        self.application =application_name
        print self.application
        self.srb_port = srb_port
        self._stopevent = threading.Event()

    def run(self):
        """ thread function to refresh the gmond data """

        while not self._stopevent.isSet():
            print "refresh gmond_data"
            self.init_obj.refresh_gmond(application = self.application, srb_port = self.srb_port)
            sleep_counter = 0
            while sleep_counter != 5:
                if self._stopevent.isSet():
                    break
                sleep(1)
                sleep_counter+=1

    def stop(self):
        """ stop the thread """
        if not self._stopevent.isSet():
            if self.i_verbose:
                print "ClientDataThread stopped"
            self._stopevent.set()
            self.join(timeout=1)
        self.init_obj.kill_gmond()



### main function of the server
```

```
def main():
    """ main function which controls the connection to the client """
    command = Cgetoptions().start()
    init_obj = Cinitialization(command['verbose'], ganglia_folder = ganglia_directory)

    application_name = None

    try:
        i_portnumber = int(command['port'])
    except ValueError:
        print "bad port given take 5000"
        i_portnumber = 5000
    serversocket = CsocketServer(i_verbose = command['verbose'], i_port = i_portnumber)
    try:
        while True:

            if init_obj.set_pid() != "":
                try:
                    init_obj.kill_gmond()
                except:
                    if command['verbose']:
                        print "gmond is not running"

            ip_address = serversocket.listen()
            i_receive_counter = 0
            i_srbport = None
            while True:

                data_received = serversocket.receive()
                #print serversocket.p_conn.getsockname()
                if command['verbose']:
                    print data_received

                if (data_received == "quit"):
                    if command['verbose']:
                        print "kill server"
                    break
                #elif(data_received.count("hostname")):


                elif (data_received.count("srbport")):

                    i_srbport = data_received[len("srbport")+1:]
                    if command['verbose']:
                        print "gotsrbport: ", i_srbport
                elif (data_received == "clientquit"):
                    if command['verbose']:
                        print "client wants to go"
                    refresh_gmond.stop()
                    break
                elif (data_received.count("application")):
                    client_hostname = data_received[len("hostname")+1:]
                    if command['verbose']:
                        print "Client hostname:", ip_address
                    init_obj.write_gmond_conf(ip_address)
                    init_obj.start_gmond()
                    application_name = data_received[(len("application")+1):]
                    if command['verbose']:
                        print "application name: ", application_name
                    refresh_gmond = RefreshGanglia(init_obj, i_srbport, application_name, i_verbose =1)
                    refresh_gmond.start()

                else:
                    if command['verbose']:
                        print "got waste from the client"
                    i_receive_counter = i_receive_counter + 1
                    if (i_receive_counter == 3):
                        break
            serversocket.clientclose()
            if (data_received == "quit"):
                try:
```

```
                    refresh_gmond.stop()
                except:
                    print "gmond refreshing thread is not running"
                break

    except KeyboardInterrupt:
        print "Server will be closed"
        serversocket.socket_close()
        if init_obj.set_pid() != "":
            try:
                refresh_gmond.stop()
            except:
                print "gmond refreshing thread is not running"
        sys.exit(0)

    serversocket.socket_close()



if __name__ == '__main__':
    sys.exit(main())
```

B.2 socket_connection.py

```
#!/usr/bin/env python

""" Performance test program (Server) to catch information about applications running on the SRB """

__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "07.09.2005"
__version__ = "0.1"
__revision__ = "1.0"

""" socket function to provide the connection between the Client and the Server process """
import socket
import sys
import struct
import select
import thread
import threading
import os
import time
i_portnumber = 5511
i_hostname = "127.0.0.1"


class CsocketServer:
    """ create a server socket to allow connections to this machine """
    def __init__(self, i_host = 'localhost', i_port = 5555, i_verbose = 0):
        """ constructor for CsocketServer class """
        self.i_host = i_host
        self.i_port = i_port
        self.i_verbose = i_verbose
        self.p_conn = 0
        try:
            if self.i_verbose:
                print 'Socket: Creating SocketServer'
            self.p_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.p_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1);
        except socket.error:
            print Exception+'Failed to create SocketServer object!!'

        try:
            if self.i_verbose:
```

```
                print 'Socket: Binding Socket'
            self.p_sock.bind((socket.gethostname(), self.i_port))
            #self.p_sock.setblocking(False)

        except socket.error:
            print "SocketError: Port may be already used or device not ready"
            self.socket_close()
            sys.exit(0)


    def listen(self, msg = 'Accepted Connection from:'):
        """ listen for client which want to connect to our server """

        if self.i_verbose:
            print 'Socket: Listening to port', self.i_port
        self.p_sock.listen(1)
        self.p_conn, i_remote_host = self.p_sock.accept()
        print self.p_sock.getsockname()
        if i_remote_host[0]:
            if self.i_verbose:
                print 'Socket: Got connection from', i_remote_host[0]
            print msg, i_remote_host[0]
        return i_remote_host[0]

    def send(self, data):
        """ send data to clients via the connection socket
        (first listen to get a connection to a clientbefore send)"""

        if self.i_verbose:
            print 'Socket: Sending data of size ', len(str(data))

        s_content = str(data)

        n_length= socket.htonl(len(s_content))

        size = struct.pack("L", n_length)

        try:
            sent = self.p_conn.send(size+s_content)
        except:

            print "SocketError: Cannot send %s" % (s_content)
            return -1

        if self.i_verbose:
            print 'Socket: Data sent!!'
        return 0



    def receive(self):
        """ receive data from the client via the connection socket
        (first listen to get a connection to a client)"""
        data_send = ""
        if self.i_verbose:
            print "Receiving length..."

        size = struct.calcsize("L")
        try:
            size = self.p_conn.recv(size)
        except:
            print "Socket: Cannot receive the length"
            return "quit"
        try:
            size = socket.ntohl(struct.unpack("L", size)[0])

        except:
            print "Socket: Bad format for send size"
            return "quit"

```

```
        data_send = data = ""
        size_decrement = size
        if self.i_verbose:
            print 'Socket: Receiving data...'
        while len(data_send) < size:
            try:
                if size_decrement < 1024:
                    receive_size = size_decrement
                else:
                    receive_size = 1024
                    size_decrement -= 1024
                data = self.p_conn.recv(receive_size)
            except Exception:
                print "SocketError: Receive Error"
                return "quit"

            data_send = data_send+data
        return data_send

    def clientclose(self):
        """ close the connection to client client (connection socket will be killed) """
#        self.engaged = 0
        self.p_conn.close()

    def socket_close(self):
        """ close the main socket to completely release the open socket """
        if self.i_verbose:
            print 'Socket: Closing socket!!'
        self.p_sock.close()
        if self.i_verbose:
            print 'Socket: Socket Closed!!'

    def __str__(self):
        """ get the port and the host from the server --- just debugging """
        return 'SocketServer:\nSocket bound to Host='+str(self.i_host)+',Port='+str(self.i_port)

class CsocketClient:
    """ connection to a server via a connected client socket """
    def __init__(self, i_remote_host = '', i_remote_port = 5000, i_verbose = 1):
        """ constructor for CsocketClient class """
        self.i_verbose = i_verbose
        self.i_remote_host = i_remote_host
        self.i_remote_port = i_remote_port
        try:
            if self.i_verbose:
                print 'Socket: Creating Socket'
            self.p_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#            self.p_sock.settimeout(5)
        except socket.error:
            print Exception+'SocketError in Socket Object Creation!!'

    def connect(self, i_remote_host = '', i_remote_port = 0):
        """ connect to a server port """
        #print i_remote_host," ",i_remote_port
        if i_remote_host == '':
            print "Socket: no host is given, try localhost"
            i_remote_host = "localhost"
        if i_remote_port == 0:
            print "no port is given, try %s" % self.i_remote_port
            i_remote_port = self.i_remote_port
        try:
            self.i_remote_host, self.i_remote_port = i_remote_host, int(i_remote_port)
        except:
            print "SocketError: Wrong parameter for host or port"
            return -1
        print "host", self.i_remote_host, "port", self.i_remote_port
        try:
            if self.i_verbose:
                print 'Socket: Connecting to '+str(self.i_remote_host)+' on port '+str(self.i_remote_port)
            self.p_sock.connect((str(self.i_remote_host), int(self.i_remote_port)))
            if self.i_verbose:
```

```
                print 'Socket: Connected !!!'
            return 0
        except socket.error:
            print "SocketError: Connection refused to %s  on port %i " % (self.i_remote_host, self.i_remote_port)
            return -1

        except KeyboardInterrupt:
            print "SocketError: Could not connect KeyboardInterrupt"
            return -1


    def send(self, data):
        """ send data to the servers via the connection socket
        (first listen to get a connection to a clientbefore send)"""

        if self.i_verbose:
            print 'Socket: Sending data of size ', len(str(data))

        s_content = str(data)

        n_length= socket.htonl(len(s_content))

        size = struct.pack("L", n_length)
        try:
             sent = self.p_sock.send(size+s_content)
        except:
            print "SocketError: Cannot send %s" % (s_content)
            return -1

        if self.i_verbose:
            print 'Socket: Data sent!!'
        return 0

    def receive(self):
        """ receive data from the client via the connection socket
        (first listen to get a connection to a client)"""
        data_send = ""
        if self.i_verbose:
            print "Receiving length..."

        size = struct.calcsize("L")
        try:
            size = self.p_sock.recv(size)
        except:
            print "Socket: Cannot receive the length"
            return "quit"
        try:
            size = socket.ntohl(struct.unpack("L", size)[0])

        except:
            print "Socket: Bad format for send size"
            return "quit"

        data_send = data = ""
        size_decrement = size
        if self.i_verbose:
            print 'Socket: Receiving data...'
        while len(data_send) < size:
            try:
                if size_decrement < 1024:
                    receive_size = size_decrement
                else:
                    receive_size = 1024
                    size_decrement -= 1024
                data = self.p_sock.recv(receive_size)
            except Exception:
                print "SocketError: Receive Error"
                return "quit"

            data_send = data_send+data
        return data_send
```

```
255
    def close(self):
        """ close the connection to the server """
        #print "close"
        self.p_sock.close()
260
    def __str__(self):
        """ get the remote host and the remote port --- just debugging"""
        return 'SocketClient\nClient connected to Host='+str(self.i_remote_host)+\
        ',Port='+str(self.i_remote_port)
```

B.3 ganglia.py

```
#!/usr/bin/env python

""" GANGLIA module to start refresh and stop the daemon """
__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
5 __date__ = "17.10.2005"
__version__ = "0.1"
__revision__ = "1.0"

""" the module os is necessary to start several programs by calling os.sytem("%program%") """
10 import os
""" sleep function is needed to give ganglia a particular amount of time to get the system-values that are needed """
from time import sleep
import re
from telnetlib import Telnet
15 import commands


class Cinitialization:
    """ Class to initialise the gmond server and to set all necessary variables """
20
    def __init__(self, i_verbose=0, ganglia_folder = None):
        """ constructor of the Cinitialization class"""
        self.pid = 0
        self.__i_verbose = i_verbose
25
        if (ganglia_folder == None):
            self.ganglia_folder = "~/programs/ganglia/"
        else:
            self.ganglia_folder = ganglia_folder
30
    def start_gmond(self,gmond_conf = "gmond_test.conf"):
        """ start the ganglia monitoring daemon """
        os.system(self.ganglia_folder+"sbin/gmond -c " \
        +self.ganglia_folder+"sbin/"+gmond_conf)
35      self.set_pid(gmond_conf)

    def set_pid(self,gmond_conf = "gmond_test.conf"):
        """ get and set the pid for the own gmond process to kill it at the end"""
        self.pid = commands.getoutput("ps -o pid,command -C gmond | grep "+gmond_conf+" | awk '{print $1}'")
40      return self.pid

#    def __no_application(self):
#        """ this function will be used if no application has send """
#        stringer = self.ganglia_folder+"""bin/gmetric -c """ \
45 #       +self.ganglia_folder+"""sbin/gmond_test.conf --name cpu_used_srb_server \
#       --value 0.0 --type float --units % """
#
#        os.system(stringer)
#        # no filedescriptors
50 #       stringer = self.ganglia_folder+"""bin/gmetric -c """ \
#       +self.ganglia_folder+"""sbin/gmond_test.conf --name number_filedesc_srb_server \
```

```
#        --value 0 --type int16 """
#
#        os.system(stringer)
#        # no memory use of srb
#        stringer = self.ganglia_folder+"""bin/gmetric -c """ \
#        +self.ganglia_folder+"""sbin/gmond_test.conf --name mem_used_srb_server \
#        --value 0.0 --type float --units Byte"""
#
#        os.system(stringer)

    def refresh_gmond(self, application = None, srb_port = None):
        """ refresh_gmond refreshes the values of Ganglia """

        if (srb_port!=None and self.__i_verbose):
            if self.__i_verbose:
                print "srb_application: ", srb_port

        os.system(self.ganglia_folder+"bin/gmetric -c " \
            +self.ganglia_folder+"sbin/gmond_test.conf --name hostname \
            --value `hostname -f` --type string ")

        # get the information for srb application of the srb_server
        if srb_port != None:
            #print """srb-port without application"""
            os.system(self.ganglia_folder+"bin/gmetric -c " \
            +self.ganglia_folder+"sbin/gmond_test.conf --name srb_fd \
            --value `./num_fd.sh "+srb_port+"` --type int16 ")

            os.system(self.ganglia_folder+"bin/gmetric -c " \
            +self.ganglia_folder+"sbin/gmond_test.conf --name srb_cpu \
            --value `./average.sh "+srb_port+"` --type float")

            stringer = self.ganglia_folder+"""bin/gmetric -c """ \
            +self.ganglia_folder+"""sbin/gmond_test.conf --name srb_mem \
            --value `./average_mem.sh """+srb_port+"""` --type float --units Byte"""
            os.system(stringer)

            stringer = self.ganglia_folder+"""bin/gmetric -c """ \
            +self.ganglia_folder+"""sbin/gmond_test.conf --name srb_cmd \
            --value srb --type float --units Byte"""
            os.system(stringer)

        i = -1

        # get the information for the normal applications
        if application != '':
            apps = application.split(';')
            if apps[-1]=='':
                del apps[-1]
            p = re.compile('^ ')
            s_app = []
            for i in range(0,len(apps)):
                #print i
                apps[i] = p.sub("",apps[i])

                f_cpu = f_mem = 0.0
                #print "appsi ",apps[i], i
                s_cpu_mem = commands.getoutput("""ps -C "%s" -o pmem,pcpu | awk '{print $1 " " $2}'""" % apps[i])
                if s_cpu_mem != "":
                    cpu_mem = s_cpu_mem.split("\n")


                    for x in cpu_mem:
                        if re.match("[0-9.]",x):

                            x = x.split(" ")
                            f_cpu += float(x[1])
                            f_mem += float(x[0])

                s = re.compile('[a-zA-Z0-9_\-\.]+')
                s_app.append(s.match(apps[i]))
```

```
                s_app[i] = s_app[i].group()
                i_count =0
125             for x in range(0,i):
                    if s_app[x].find(s_app[i]) != -1:
                        i_count +=1
                if i_count != 0:
                    print "not two commands with the same application"
130                 continue


                stringer = self.ganglia_folder+"""bin/gmetric -c """ \
                +self.ganglia_folder+"""sbin/gmond_test.conf --name  """+s_app[i]+"""_mem \
135             --value "%s" --type float --units Byte""" % f_mem


                """ set the used file_descriptor value of the srb_server """
                l_pid = []
140             s_pid = commands.getoutput("ps u -C '"+apps[i]+"' | grep $USER | awk '{print $2}'");

                #""" just the open application of the user will be measured"""
                l_pid = s_pid.split("\n");
                #""" divide the different application pids"""
145             i_number_fd = 0
                for s_single_pid in l_pid[0:]:
                    try:

                        s_number_fd = commands.getoutput("ls -l /proc/"+s_single_pid+"/fd | wc -l")
150                     """ get the number of open file descriptors"""
                        i_number_fd += int(s_number_fd)-1
                        """ the header should not be counted """
                    except ValueError:
                        print "Programme %s cannot be monitored!" % apps[i]
155                     break

                stringer = stringer+";"+self.ganglia_folder+"bin/gmetric -c " \
                +self.ganglia_folder+"sbin/gmond_test.conf --name "+s_app[i]+"_fd --type int16  \
                --value "+str(i_number_fd)
160

                stringer = stringer+";"+self.ganglia_folder+"""bin/gmetric -c """ \
                +self.ganglia_folder+"""sbin/gmond_test.conf --name """+s_app[i]+"""_cpu \
                --value "%s" --type float --units Byte""" % f_cpu

165



                stringer = stringer+";"+self.ganglia_folder+"""bin/gmetric -c """ \
                +self.ganglia_folder+"""sbin/gmond_test.conf --name """+s_app[i]+"""_cmd \
                --value "%s" --type float --units Byte""" % apps[i]
170

                os.system(stringer)

        if i != -1:

175             i+=1
                if srb_port != None:
                    i += 1
                stringer = self.ganglia_folder+"""bin/gmetric -c """ \
                +self.ganglia_folder+"""sbin/gmond_test.conf --name num_of_apps \
180             --value "%s" --type float --units Byte""" % i

                os.system(stringer)
        #else:
         #    self.__no_application()
185     #sleep that gmond can catch all the variables
        sleep(0.1)


    def kill_gmond(self):
190     """debug: kill gmond before start it new"""
        if self.pid == 0:
            self.set_pid()
        print self.pid
```

```
        if self.pid != "":
            try:
                commands.getoutput("kill -9 "+str(self.pid));
            except:
                print "gmond daemon already killed"
        """ wait a 0.2 second until we can restart the gmond daemon"""
        sleep(0.2)

    def get_gmond_output(self):
        """ catch the XML output of gmond """
        tn = Telnet('localhost', 8649)   # connect to finger port
        telnet_data = tn.read_all()
        tn.close()
        return telnet_data

    def write_gmond_conf(self,client_hostname):
        """ create a new gmond conf, which says where ganglia has to send the data """
        output = open(os.path.expanduser(self.ganglia_folder+"sbin/gmond_test.conf"),'w')
        output.write("""/* This configuration is as close to 2.5.x default behavior as possible
   The values closely match ./gmond/metric.h definitions in 2.5.x */
globals {
  setuid = no
/*  user = nobody              */
  cleanup_threshold = 300 /*secs */
}

/* If a cluster attribute is specified, then all gmond hosts are wrapped inside
 * of a <CLUSTER> tag.  If you do not specify a cluster tag, then all <HOSTS> will
 * NOT be wrapped inside of a <CLUSTER> tag. */

/* Feel free to specify as many udp_send_channels as you like.  Gmond
   used to only support having a single channel */

udp_send_channel {
    host = %s
    port = 8649
  }

/* You can specify as many udp_recv_channels as you like as well. */


/* You can specify as many tcp_accept_channels as you like to share
   an xml description of the state of the cluster */



/* The old internal 2.5.x metric array has been replaced by the following
   collection_group directives.  What follows is the default behavior for
   collecting and sending metrics that is as close to 2.5.x behavior as
   possible. */

/* This collection group will cause a heartbeat (or beacon) to be sent every
   20 seconds.  In the heartbeat is the GMOND_STARTED data which expresses
   the age of the running gmond. */

/* This collection group will send general info about this host every 1200 secs.
   This information doesn't change between reboots and is only collected once. */


/*collection_group {
  collect_once = yes
  time_threshold = 90

} */

/* This collection group will send the status of gexecd for this host every 300 secs */
/* Unlike 2.5.x the default behavior is to report gexecd OFF.  */

/* This collection group will collect the CPU status info every 20 secs.
   The time threshold is set to 90 seconds.  In honesty, this time_threshold could be
   set significantly higher to reduce unneccessary network chatter. */
```

```
collection_group {
  collect_every = 10
  time_threshold = 10
  /* CPU status */
  metric {
    name = "cpu_speed"
    value_threshold = "1.0"
  }
  metric {
    name = "mem_total"
    value_threshold = "1.0"
  }
  metric {
    name = "machine_type"
    value_threshold = "1.0"
  }
  metric {
    name = "cpu_user"
    value_threshold = "1.0"
  }
  metric {
    name = "cpu_nice"
    value_threshold = "1.0"
  }
  metric {
    name = "cpu_system"
    value_threshold = "1.0"
  }
  metric {
    name = "cpu_idle"
    value_threshold = "1.0"
  }
  metric {
    name = "proc_total"
    value_threshold = "1.0"
  }
  metric {
    name = "load_one"
    value_threshold = "1.0"
  }
  /* The next two metrics are optional if you want more detail...
     ... since they are accounted for in cpu_system.
  */
}


/* This group collects the number of running and total processes */

/* This collection group grabs the volatile memory metrics every 40 secs and
   sends them at least every 180 secs.  This time_threshold can be increased
   significantly to reduce unneeded network traffic. */

/* Different than 2.5.x default since the old config made no sense */ """ % client_hostname)
```

B.4 python_client.py

```
#!/usr/bin/env python

"""
* Python client to set and get gmond output from servers
*
* sip04ck
*
"""
```

```
__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "07.09.2005"
__version__ = "0.1"
__revision__ = "1.0"

import commands
""" start application """
import xml.dom.minidom
""" minidom is used to parse the ganglia-gmond-xml-file """
import sqlite
""" sqlite --> database connection between sqlite and python (pysqlite) """
import os
""" os is used for shell commands"""
import sys
""" sys is used to start shell commands and get a return value """
import threading
""" threading is used to start a singular thread to every server """
from socket_connection import CsocketClient
""" socket connection is provided by the socket_connection file """
import ConfigParser
""" the config.ini will be parsed with the Config Parser class """
import time
""" the time class is used to create timestamps """
import cpu_load
""" pattern matching """
import re

from ganglia import Cinitialization
import socket

### change here to setup another ganglia folder
ganglia_directory = "~/programs/ganglia/"
gmond_conf_file = "gmond_client.conf"

class Ctimestamp:
    """ set a timestamp format which will be stored in the database """
    def __init__(self):
        """ initialize the timestamp format """
        self.__format = '%Y-%m-%d %H:%M:%S'

    def get_timestring(self):
        """ get the time-string format """
        return time.strftime(self.__format)

    def get_timetuple(self, time_string):
        """ get the time as a tuple """
        return time.strptime(time_string, self.__format)

class Cconfigparser:
    """ parse the config.ini file to get the main arguments"""
    def __init__(self, i_verbose = 0):
        """ initialize the config dictionary """
        self._ConfigDefault = {
        "server.quantity":          1,
        "test.application":         "grep",
        "server.port_1":            5000,
        "server.host_1":            "localhost",
        "project.name":             "test",
        "project.desc":             "none",
        "test.name":                "test",
        "test.desc":                "none",
        "tester.forename":          "Carsten",
        "tester.surname":           "Koebernick",
        "tester.email":             "c.koebernick@rl.ac.uk",
        "measurement.quantity":      10,
        "measurement.application":  "bash",
        "measurement.srb":           0,
        "measurement.poll_time":     15,

        }
        self.__config_file = None
```

```
80          self.__i_verbose = i_verbose


        def load_config(self, config_file):
            """ parse function to extract the information of the config.ini """
85          if config_file == None:
                config_file = "config.ini"
            self.__config_file = config_file

            if self.__i_verbose:
90              print "The config-file: %s" % (self.__config_file)

            config = self._ConfigDefault.copy()
            try:
                cp_object = ConfigParser.ConfigParser()
95              cp_object.read(config_file)
                if cp_object.sections() == []:
                    print "config file does not exist or is empty, take built in config"
                else:
                    for section in cp_object.sections():
100                     name = section.lower()
                        for option in cp_object.options(section):
                            config[name + "." + option.lower()] = cp_object.get(section, option).strip()
                if self.__i_verbose:
                    print "config-file successfully scanned"
105         except Exception:
                print "bad config file - use standard config"
                print "Shall I write the standard config to "+config_file+ " ? (y/anykey)"
                s_input =""
                s_input = sys.stdin.read(1)
110             if s_input == 'y':
                    self.write(config_file, config)
                if self.__i_verbose:
                    print "take standard config for the application"
            return config
115

        def get_config_file(self):
            """ send the config_file name """
            return self.__config_file

120     def write(self, file_name, config = None):
            """
            given a dictionary with key's of the form 'section.option: value'
            write() generates a list of unique section names
            creates sections based that list
125         use config.set to add entries to each section
            """
            if config == None:
                config = self._ConfigDefault
            try:
130             file_desc = open(file_name, 'w')
            except IOError:
                print "Cannot open the file for writing: permission denied"
                return -1
            conf_parser_obj = ConfigParser.ConfigParser()
135         #a little string hacking because our section names are un-normalized
            #this builds a list of all the sections names
            sectionslst = []
            sections = []
            for k in config.keys():
140             sectionslst.append(k.split(".")[0])
            #get unique entries

            sections = self.__uniquer(sectionslst)
            for sec in sections:
145             #make the headers
                conf_parser_obj.add_section(sec)
                #for each item in dictionary
                #it splits the key in two and uses that for the first and second "set" args
                #then it uses the item.value for the 3rd arg
150             # from 'section.option:value'
```

```
        for k in config.items():
            conf_parser_obj.set( k[0].split(".")[0], k[0].split(".")[1], k[1] )
        conf_parser_obj.write(file_desc)
        file_desc.close()
        if self.__i_verbose:
            print "written to ", file_name
        return 0

    def __uniquer(self, seq):
        """ get unique entries """

        seen = {}
        result = []

        for item in seq:
            if item in seen:
                continue
            seen[item] = 1
            result.append(item)
        return result


class Cdatabase:
    """ database class provides some essential sqlite functions like connect, create and check database """
    def __init__(self, db = None, i_verbose = 0):
        """ constructor for database class : test if database is available, if not create one """
        self.__i_verbose = i_verbose
        self.__wait_for_all_measurements = threading.Event()
        self.d_measurement_table = {
        "is_id":                    "INTEGER",
        "time":                     "VARCHAR(30)",
        "server_cpu_system":        "VARCHAR(10)",
        "client_cpu_system":        "VARCHAR(10)",
        "server_cpu_user":          "VARCHAR(10)",
        "server_cpu_idle":          "VARCHAR(10)",
        "server_cpu_nice":          "VARCHAR(10)",
        "server_proc_total":        "VARCHAR(10)",
        "client_cpu_system":        "VARCHAR(10)",
        "client_cpu_user":          "VARCHAR(10)",
        "client_cpu_idle":          "VARCHAR(10)",
        "client_cpu_nice":          "VARCHAR(10)",
        "client_proc_total":          "VARCHAR(10)",
        }

        self.d_application_table = {
            "name":                 "VARCHAR(30)",
            "memory":               "VARCHAR(10)",
            "cpu":                  "VARCHAR(10)",
            "fd":                   "VARCHAR(10)",
            "measurement_id":       "INTEGER",
            "command":              "VARCHAR(100)",
            }

        self.__l_measurements = []
        if (db == None):
            self.__db = "./srb.db"
        else:
            self.__db = db
        self.__timestamp = Ctimestamp()

        if (self.__check_database__() == 0):
            if self.__i_verbose:
                print "create a new database"
            self.db_con = 0

            print "Database file not existent, create a new database: "+self.__db+" ?"
            while 1:
                try:
                    user_answer = raw_input("(Y/n)?")
                    if user_answer == "y" or user_answer == "Y" or user_answer == "":
                        self.__create_database__()
```

```
                    break
                elif user_answer == "n":
                    print "Programme stops"
                    sys.exit(0)
            except KeyboardInterrupt:
                print "Programme stops"
                sys.exit(0)


        else:
            if self.__i_verbose:
                print "take the existent datbase srb.db"
            self.db_con = sqlite.connect(self.__db, autocommit=1);
            self.db_cursor = self.db_con.cursor();
            try:
                self.db_cursor.execute("""SELECT * FROM project""")
            except:
                print "bad db file"
                sys.exit(-1)
        self.__lock = threading.Lock()


    def set_new_db(self,db):

        self.db_cursor.close()
        self.db_con.close()
        self.__db = db
        if (self.__check_database__() == 0):
            if self.__i_verbose:
                print "create a new database"
            self.db_con = 0;
            self.__create_database__()

        else:
            if self.__i_verbose:
                print "take the existent datbase srb.db"
            try:
                self.db_con = sqlite.connect(self.__db, autocommit=1);
                self.db_cursor = self.db_con.cursor();

            except sqlite.Error:
                print "cannot open the db"
                return -1
            try:
                self.db_cursor.execute(""" SELECT * FROM project """)
                self.db_cursor.fetchall()
                return 0
            except sqlite.DatabaseError:
                print "bad database"
                return -1

    def _get_db(self):
        return self.__db

    def __create_database__(self):
        """ database creation if the srb.db file is not there """

        self.db_con = sqlite.connect(self.__db, autocommit=1);

        self.db_cursor = self.db_con.cursor()


        self.db_cursor.execute('CREATE TABLE project (project_id INTEGER PRIMARY KEY, title VARCHAR(100) UNIQUE,
            description TEXT)')

        self.db_cursor.execute('CREATE TABLE tester (tester_id INTEGER PRIMARY KEY, surname VARCHAR(30), forename
            VARCHAR(30), email VARCHAR(100))')

        self.db_cursor.execute('CREATE TABLE test (test_id INTEGER PRIMARY KEY, name VARCHAR(100), description TEXT, \
                                failed_flag INTEGER, tester_id INTEGER, project_id INTEGER)')

```

```
        self.db_cursor.execute('CREATE TABLE host (host_id INTEGER PRIMARY KEY, hostname VARCHAR(100) UNIQUE,
            ip_address VARCHAR(20), \
                                cpu_speed INTEGER, memory INTEGER)')

        self.db_cursor.execute('CREATE TABLE srbserver (srbserver_id INTEGER PRIMARY KEY, srb_port INTEGER, host_id
            INTEGER )')

        self.db_cursor.execute('CREATE TABLE iteration_srbserver(is_id INTEGER PRIMARY KEY, iteration_id INTEGER,
            srbserver_id INTEGER)')

        self.db_cursor.execute('CREATE TABLE iteration (iteration_id INTEGER PRIMARY KEY, test_id INTEGER, time
            VARCHAR(30), application VARCHAR(100))')

        s_meas_create = s_app_create = ""
        for key, value in self.d_measurement_table.items():
            s_meas_create += " %s %s," %(key,value)

        self.db_cursor.execute('CREATE TABLE measurement (measurement_id INTEGER PRIMARY KEY, %s )' % (s_meas_create
            [:-1]))

        for key, value in self.d_application_table.items():
            s_app_create += " %s %s," %(key,value)

        self.db_cursor.execute('CREATE TABLE application (application_id INTEGER PRIMARY KEY, %s )' % (s_app_create
            [:-1]))




        if self.__i_verbose:
            print "database created"


    def __check_database__(self):
        """ check if the file accessable """
        access_flag = os.access(self.__db, os.F_OK);
        return access_flag


    def lock_it(self):
        """ lock the datbase """
        self.__lock.acquire()
        if self.__i_verbose:
            print "database locked"


    def rlock(self):
        """ release the lock from the database """
        self.__lock.release()
        if self.__i_verbose:
            print "database released"


    def insert_measurement (self, d_xmlstream, is_id,clientload):
        """ insert a measurement into the database """

        time_stamp = self.__timestamp.get_timestring()

        self.d_measurement_table = {
        "is_id":                        is_id,
        "time":                         "'%s' " % time_stamp,
        "server_cpu_system":            float(d_xmlstream["cpu_system"]["VAL"]),
        "server_cpu_user":              float(d_xmlstream["cpu_user"]["VAL"]),
        "server_cpu_idle":              float(d_xmlstream["cpu_idle"]["VAL"]),
        "server_cpu_nice":              float(d_xmlstream["cpu_nice"]["VAL"]),
        "server_proc_total":            float(d_xmlstream["proc_total"]["VAL"]),
        "client_cpu_system":            float(clientload["cpu_system"]["VAL"]),
        "client_cpu_user":              float(clientload["cpu_user"]["VAL"]),
        "client_cpu_idle":              float(clientload["cpu_idle"]["VAL"]),
        "client_cpu_nice":              float(clientload["cpu_nice"]["VAL"]),
```

```
        "client_proc_total":            float(clientload["proc_total"]["VAL"]),
        }
        key_match = re.compile("(.*)_(fd|mem|cmd|cpu)$")
        key_found = {}

        s_keys=s_items =""
        for key,item in self.d_measurement_table.items():
            s_keys += " %s," % key
            s_items += " %s," % item

        self.db_cursor.execute("""INSERT INTO measurement ( %s ) VALUES ( %s )""" % (s_keys[:-1],s_items[:-1]))

        self.db_cursor.execute("""SELECT M.* from measurement AS M, test AS T where M.is_id = '%s' AND M.time = '%s' "
            "" %(is_id, str(time_stamp)))
        measurements = self.db_cursor.fetchall()
        for key in d_xmlstream.keys():
            if key_match.match(key):
                key = re.sub("_(fd|mem|cmd|cpu)$","",key)
                if d_xmlstream.has_key(key+"_fd") and d_xmlstream.has_key(key+"_cpu") \
                and d_xmlstream.has_key(key+"_mem") and d_xmlstream.has_key(key+"_cmd") and not key_found.has_key(key)
                    :
                    key_found[key] = 1

                    self.d_application_table = {
                    "name":                     "'%s'" % key,
                    "memory":                   "'%s'" % d_xmlstream[key+"_mem"]["VAL"],
                    "cpu":                      "'%s'" % d_xmlstream[key+"_cpu"]["VAL"],
                    "fd":                       "'%s'" % d_xmlstream[key+"_fd"]["VAL"],
                    "command":                  "'%s'" % d_xmlstream[key+"_cmd"]["VAL"],
                    "measurement_id":           measurements[0]["M.measurement_id"],
                    }
                    s_keys=s_items =""
                    for key,item in self.d_application_table.items():
                        s_keys += " %s," % key
                        s_items += " %s," % item
                    self.db_cursor.execute(""" INSERT INTO application ( %s ) VALUES ( %s ) """ % (s_keys[:-1],s_items
                        [:-1]) )

        measurements.reverse()
        self.__l_measurements.append(measurements[0])

        if self.__i_verbose:
            print "measurement saved in database"
    def set_all_measurements_there(self,set_var = None):
        if set_var == None:
            if self.__i_verbose:
                print "clear"
            self.__wait_for_all_measurements.clear()
        else:
            self.__wait_for_all_measurements.set()
            if self.__i_verbose:
                print "set"

    def get_measurements(self):
        """ return the last insert measurements"""
        if not self.__wait_for_all_measurements.isSet():
            return []
        print_measurements = self.__l_measurements
        self.__l_measurements =[]
        return print_measurements

    def insert_srbserver(self, d_xmlstream, srbport=1):
        """ insert a new srbserver to the database """

        if (d_xmlstream["hostname"]["VAL"] == None or len(d_xmlstream["hostname"]["VAL"]) == 0 ):
            print "the hostname is needed to get the srbserver_id"
            print " cannot insert the srbserver "
            return -1
        else:

            select_cmd = """ SELECT host_id FROM host WHERE hostname = '%s' """ % d_xmlstream["hostname"]["VAL"]
```

```
            self.db_cursor.execute(select_cmd)
            host_id = self.db_cursor.fetchone()
            if host_id == None:
                self.db_cursor.execute(""" INSERT INTO host (hostname,cpu_speed,memory,ip_address) VALUES ('%s','%s
                    ','%s','%s')""" % \
                                       (d_xmlstream["hostname"]["VAL"], d_xmlstream["cpu_speed"]["VAL"], d_xmlstream["
                                           mem_total"]["VAL"],d_xmlstream["IP"]))
                self.db_cursor.execute(select_cmd) # can be replaced with host_id = self.db_cursor.lastrowid
                host_id = self.db_cursor.fetchone()

            select_cmd = """ SELECT srbserver_id FROM srbserver WHERE srb_port = '%s' and host_id = '%s' """ % (
                srbport, host_id[0])
            self.db_cursor.execute(select_cmd)  # can be replaced with srbserver_id = self.db_cursor.lastrowid
            srbserver_id = self.db_cursor.fetchone()

            if (srbserver_id == None):
                #print "srb:", srbport
                self.db_cursor.execute("""INSERT INTO srbserver (srb_port, host_id) VALUES \
                                       ('%s','%s')""" % (srbport, host_id[0]))

                self.db_cursor.execute(select_cmd)
                srbserver_id = self.db_cursor.fetchone()
                if self.__i_verbose:
                    print "try to insert a srbserver with hostname %s" % (d_xmlstream["hostname"]["VAL"])
            else:
                if self.__i_verbose:
                    print "found the hostname: %s of the srbserver with the id: %s" % (d_xmlstream["hostname"]["VAL"],
                        srbserver_id[0])
            return srbserver_id[0]


    def insert_iteration_srbserver(self, iteration_id, srbserver_id):
        """ insert a new iteration_srbserver combination to the database """

        select_cmd = """ SELECT is_id FROM iteration_srbserver WHERE iteration_id = %s and srbserver_id = %s """ % (
            iteration_id, srbserver_id)

        self.db_cursor.execute(select_cmd)
        is_id = self.db_cursor.fetchone()
        #print tester_id+" "+project_id+" "
        if (is_id == None):
            self.db_cursor.execute("""INSERT INTO iteration_srbserver (iteration_id, srbserver_id) VALUES (%d, %d)"""
                % (iteration_id, srbserver_id) )

            # might be replaced with is_id = self.db_cursor.lastrowid
            self.db_cursor.execute(select_cmd)
            is_id = self.db_cursor.fetchone()

            if self.__i_verbose:
                print "insert a new srbserver: %d and test: %d combination in the iteration_srbserver table" % (
                    srbserver_id, iteration_id)
        return is_id[0]


    def insert_project(self, s_project_title, s_project_desc):
        """insert a new project"""

        project_id = self.db_cursor.execute("""SELECT project_id from project WHERE title = '%s'""" % (s_project_title
            ))
        project_id = self.db_cursor.fetchone()


        try:
            if (project_id == None):

                self.db_cursor.execute("""INSERT INTO project (title, description) VALUES ('%s', '%s')""" % (
                    s_project_title, s_project_desc))

                self.db_cursor.execute("""SELECT project_id FROM project WHERE title = '%s'""" % (s_project_title))
        # can be replaced with project_id = self.db_cursor.lastrowid
```

```
                project_id = self.db_cursor.fetchone()

                if self.__i_verbose:
                    print "new project: %s inserted in database with the project_id: %s" % (s_project_title,
                        project_id[0])
490
            return project_id[0]

        except Exception:
            print Exception
495         print "bad config file options cannot check the project_id"
            return -1


    def insert_test(self, s_test_title, s_test_desc, project_id, tester_id, s_application):
500     """insert a new test"""

        select_cmd = """ SELECT test_id FROM test WHERE project_id = %d AND name = '%s'""" % (project_id, s_test_title
            )
        self.db_cursor.execute(select_cmd)
        test_id = self.db_cursor.fetchone()
505     if (test_id == None):

            self.db_cursor.execute("""INSERT INTO test (name, description, project_id, tester_id, failed_flag) VALUES
                \
                                    ( '%s' , '%s', %d, %d ,%d)""" % (s_test_title, s_test_desc, project_id,
                                        tester_id, 0))

510         self.db_cursor.execute("""SELECT test_id FROM test WHERE name = '%s' AND project_id = %i""" % (
                s_test_title, project_id))
            test_id = self.db_cursor.fetchone() # can be replaced with test_id = self.db_cursor.lastrowid
            if self.__i_verbose:
                print "new test: %s inserted in database with the test_id: %s" % (s_test_title, test_id[0])
        time_stamp = self.__timestamp.get_timestring()
515     self.db_cursor.execute(""" INSERT INTO iteration (test_id , time, application) VALUES (%d ,'%s', '%s') """ % (
            test_id[0], time_stamp, s_application))
        self.db_cursor.execute("""SELECT iteration_id FROM iteration WHERE test_id = %d AND time = '%s'""" % (test_id
            [0], time_stamp))
        iteration_id = self.db_cursor.fetchone() # can be replaced with iteration_id = self.db_cursor.lastrowid
        return iteration_id[0]

520

    def insert_tester(self, s_tester_forename, s_tester_surname, s_tester_email):
        """ insert a tester to the database and return the id """

        if (len(s_tester_surname) == 0 or len(s_tester_forename)==0):                                   #obsolete
525         print "the surname and the forename are needed to put the tester in the database"
            sys.exit(-1)
        else:
            select_cmd = """ SELECT tester_id FROM tester WHERE surname = '%s' AND forename = '%s'""" % (
                s_tester_surname, s_tester_forename)
            self.db_cursor.execute(select_cmd)
530         tester_id = self.db_cursor.fetchone()

            if (tester_id == None):
                self.db_cursor.execute("""INSERT INTO tester(surname, forename, email) VALUES\
                                    ('%s', '%s', '%s')""" % (s_tester_surname, s_tester_forename, s_tester_email))
535
                self.db_cursor.execute(select_cmd)
                tester_id = self.db_cursor.fetchone()# can be replaced with tester_id = self.db_cursor.lastrowid
                if self.__i_verbose:
                    print "new tester: %s,%s inserted in database with tester_id: %s" % (s_tester_surname,
                        s_tester_forename, tester_id[0])
540         return tester_id[0]


    def select_table(self, s_table_name, s_column, s_where_column = None, s_where_content = None):
        """ select all from a table and a particular row """
545
        if (s_where_column != None):
```

```
            s_select = """ SELECT %s from %s where %s = %s""" % (s_column, s_table_name, s_where_column,
                 s_where_content)
        else:
            s_select = """ SELECT %s from %s""" % (s_column, s_table_name)
        self.db_cursor.execute(s_select)
        select_vars = self.db_cursor.fetchall()
        if self.__i_verbose:
            print "get table: %s " % (s_select)
        return select_vars


    def select_measurement(self, s_columns, d_where_column):
        """ select a particular measurement, depending on time, test-id or srbserver-id """

        s_select_part = ""
        for d_column in d_where_column:
            s_select_part += d_column['column']+ d_column['innerconnect']+ d_column['content']+ " "+d_column['connect'
                ]

        s_select = "Select %s from measurement where %s" % ( s_columns, s_select_part)

        self.db_cursor.execute(s_select)
        select_vars = self.db_cursor.fetchall()
        if self.__i_verbose:
            print "get measurement: %s " % (s_select)
        return select_vars

class ServerConnect:
    """ connects the client to the running server to send the application names """

    def __init__(self, i_port = '', s_host = '', i_quantity = 0, \
                 s_application = "bash", i_server = 1, srbport = None, i_verbose = 0):
        self.__i_verbose = i_verbose
        self.__clientsocket = CsocketClient(i_verbose)
        if s_host == "127.0.0.1" or s_host == "localhost":
                    print "Please use real hostnames or IP-addresses no localhost or 127.0.0.1"
                    self.connected = -1
        else:
            self.connected = self.__clientsocket.connect(i_remote_host = s_host, i_remote_port = i_port )
        if self.__i_verbose:
            print "Server %s connection status: %s" % (i_server, self.connected)
        if self.connected != -1:
            self.__srbport = srbport
            if self.__srbport != None:

                if self.__i_verbose:
                    print "Send the srb_port, because a srb_application will be measured"
                data = "srbport %s" % (str(srbport))
                self.__clientsocket.send(data)
            self.__application = s_application

            self.__i_server = i_server


    def stop_connection(self):
        """ send the server a clientquit to stop the connection """
        if self.__i_verbose:
            print "Stop connection to Server %s" % (self.__i_server)
        self.__clientsocket.send(data = "clientquit")
        self.__clientsocket.close()

    def connect(self):
        """ conect to the server, to send the application which should be measured"""
        data = "application "
        if self.__application == "none" or self.__application == "":
            if self.__srbport == None:
                data += "bash"

                #if self.__i_verbose:
                 #   print "send: %s to Server: %s" % (data, self.i_server)
        else:
```

```
            data += self.__application
            #self.clientsocket.send(data)
        if self.__i_verbose:
            print "send: %s to Server: %s" % (data, self.__i_server)
        self.__clientsocket.send(data)


"""************************** GANGLIA QUERY THREAD **************************"""

class GangliaThread(threading.Thread):
    """ class to query the ganglia with telnet, furthermore the content of Ganglia will be stored in SQLite"""

    def __init__(self, database,srbport, iteration_id, poll_time,quantity, measurement_app, i_verbose):
        """ Constructor which sets the poll_time of gmond (15sec at least) and the number of queries to ganglia (at
            least 2)"""
        self.__i_verbose = i_verbose
        if int(poll_time) > 15:
            self.__poll_time = int(poll_time)
        else:
            self.__poll_time = 15
        self.__quantity = quantity
        if self.__quantity < 2:
            self.__quantity = 2
        threading.Thread.__init__(self, name = "Refresh_gmond")
        self._stopevent = threading.Event()
        if srbport != None:
            self.__srb_port = srbport
        else:
            self.__srb_port = 1
        self.__run_app_thread = None
        self.__iteration_id = iteration_id
        self.__measurement_app = measurement_app
        self.__db_obj = database
        self.__ganglia = Cinitialization(self.__i_verbose,ganglia_folder = ganglia_directory)
        self.__parse_object = Cxmlparser()
        self.__db_obj.set_all_measurements_there()
        #self.setDaemon(True)

    def run(self):
        """ Thread which polls Ganglia"""
        self.__ganglia.start_gmond(gmond_conf=gmond_conf_file)
        time.sleep(10)
        application_started = 0
        counter = 0
        srbserverid_set = {}
        last_insert = {}
        #no_external_data = 0
        while not self._stopevent.isSet():
            s_xml_content = self.__ganglia.get_gmond_output()
            i_xml_start = s_xml_content.find("<GANGLIA_XML");
            i_xml_end = s_xml_content.find("</GANGLIA_XML>")+14;
            d_xml = self.__parse_object.parse_string(s_xml_content[i_xml_start:i_xml_end])
            host_count =0

            for machine in d_xml.keys():
                if machine == 'localhost':
                    continue
                if not srbserverid_set.has_key(machine):

                    if d_xml[machine].has_key("cpu_speed") and d_xml[machine].has_key("hostname"):
                        self.__db_obj.lock_it()
                        srbserver_id = self.__db_obj.insert_srbserver(d_xml[machine],self.__srb_port)
                        srbserverid_set[machine] = srbserver_id
                        self.__db_obj.rlock()
                    else:
                        if self.__i_verbose:
                            print "no data there for machine:",machine
                        if last_insert.has_key(machine):
                            if last_insert[machine] == d_xml[machine]["REPORTED"]:
                                print "No fresh data from Ganglia for %s, maybe server lost" % machine
                        break
```

```
                self.__db_obj.lock_it()
                is_id = self.__db_obj.insert_iteration_srbserver(self.__iteration_id, srbserverid_set[machine])

                if last_insert.has_key(machine):
                    if last_insert[machine] == d_xml[machine]["REPORTED"]:
                        print "No fresh data from Ganglia for %s, maybe server lost" % machine
                last_insert[machine] = d_xml[machine]["REPORTED"]
                host_count +=1
                self.__db_obj.insert_measurement(d_xml[machine], is_id ,d_xml['localhost'])
                self.__db_obj.rlock()
            if host_count==len(d_xml.keys())-1:
                self.__db_obj.set_all_measurements_there("set")
                if (self.__measurement_app != "none" and self.__measurement_app != "") and application_started == 0:
                    self.__run_app_thread = Crunapp(self.__measurement_app, self.__i_verbose)
                    self.__run_app_thread.start()
                    if self.__i_verbose:
                        print "Test application started!!"
                    application_started = 1

            counter += 1

            if counter > int(self.__quantity):
                if self.__i_verbose:
                    print "stop the measurement"
                self.stop()
                time.sleep(3)
                break
            sleep_counter =0
            while sleep_counter != self.__poll_time:
                time.sleep(1)
                if self._stopevent.isSet():
                    break
                sleep_counter +=1
            if self._stopevent.isSet():
                break

    def stop(self):
        """ Stop the Ganglia Query Thread """

        if not self._stopevent.isSet():
            if self.__i_verbose:
                print "GangliaThread stopped"

            self._stopevent.set()
            #threading.Thread.join(self, 1)
            if self.__run_app_thread != None:
                self.__run_app_thread.stop()
        self.__ganglia.kill_gmond()


""" ******************** XML PARSER ********************************** """
class Cxmlparser:
    """ XML parsing class with two functions, which parses the string and change the xml string into a dictionary"""

    def __init__(self, i_verbose = 0):
        """ constructor for the Cxmlparser class """
        self.__i_verbose = i_verbose
        self.__doc = ""
        self.__d_xmlstream = {}
        self.__s_xml = ""


    def parse_string(self, s_xml):
        """ parse the string start the xml-string to dictionary function"""
        self.__d_xmlstream = {}
        self.__s_xml = s_xml
        if self.__i_verbose:
            print "XML-String:\n%s" % (s_xml)
        self.__doc = xml.dom.minidom.parseString(self.__s_xml)
        self.__explore_childs()
```

```
        return self.__d_xmlstream


    def __explore_childs(self):
        """ get the xml_stream in a dictionary (recursive function)"""

        nodelist = self.__doc.childNodes
        for subnode in nodelist:
            if (subnode.nodeType == subnode.ELEMENT_NODE):
                if (subnode.tagName == "HOST"):

                    self.host = subnode.getAttribute("NAME")
                    print "host",self.host
                    self.__d_xmlstream[self.host] = {}
                    self.__d_xmlstream[self.host]["IP"]=subnode.getAttribute("IP")
                    self.__d_xmlstream[self.host]["REPORTED"]=subnode.getAttribute("REPORTED")
                if (subnode.tagName == "METRIC"):
                    self.__d_xmlstream[self.host][subnode.getAttribute("NAME")] = {}
                    self.__d_xmlstream[self.host][subnode.getAttribute("NAME")]['VAL'] = subnode.getAttribute("VAL")
                    self.__d_xmlstream[self.host][subnode.getAttribute("NAME")]['TYPE'] = subnode.getAttribute("TYPE")
                    self.__d_xmlstream[self.host][subnode.getAttribute("NAME")]['UNITS'] = subnode.getAttribute("UNITS
                        ")
                self.__doc = subnode
                self.__explore_childs()


""" ****************** MEASUREMENT CLASS ********************************** """
class CMeasurement:
    """ measurement class which is able to start and stop the measurement and insert the data in the database """

    def __init__(self, database_file, i_verbose = 0):
        """ constructor to initialize the essential values like the config file, the thread and the database """

        self.__i_verbose = i_verbose
        self.__servcon_obj = {}
        self.db_object = Cdatabase(database_file, self.__i_verbose)
        self.__d_config = ""


    def set_config(self, config):
        """ set config for measurement """
        if self.__i_verbose:
            print "set a new config: \n%s" % (config)
        self.__d_config = config


    def get_config(self):
        """ return the config """
        return self.__d_config

    def stop_connections(self):
        """ stop every open connection to a server"""

        range_var = int(self.__d_config["server.quantity"])
        #print range_var
        for i_server in range(range_var):
            if self.__servcon_obj.has_key(i_server):
                if self.__servcon_obj[i_server].connected != -1:
                    if self.__i_verbose:
                        print "close the connection of Server: %s" % (i_server)
                    self.__servcon_obj[i_server].stop_connection()


    def run_measurements(self):
        """ run the measurements in dependency of the amount of measurements"""


        i_exit_var = 0
        i_srbport = None
        for i_server in range(int(self.__d_config["server.quantity"])):
```

```
            """ start as much server as it is set in server.quantity """
            s_server_id = str(i_server + 1)
            try:
                #s_server_poll_time = int(self.__d_config["server.poll_time_"+s_server_id])
                i_port = self.__d_config["server.port_"+s_server_id]
                s_host = self.__d_config["server.host_"+s_server_id]
            except KeyError:
                print "Wrong parameter in ini-file"
                return -1
            except Exception:
                print "bad parameter set poll_time,host and port"
                return -1
            if self.__d_config["measurement.srb"] == "1":
                i_srbport = self.__get_srbport()

            if self.__i_verbose:
                print "initialise Thread for Server: %s" % (i_server)
            self.__servcon_obj[i_server] = ServerConnect(i_port = i_port, \
                                                s_host = s_host,  \
                                                i_quantity = self.__d_config["measurement.quantity"], \
                                                s_application = self.__d_config["test.application"], \
                                                i_server = i_server,srbport = i_srbport, i_verbose = self.
                                                    __i_verbose)

            """ create the thread_objects """
            if ( self.__servcon_obj[i_server].connected == -1):
                """ test if the server can be connected """
                print "The server number %s cannot be connected" % (s_server_id)
                i_exit_var = 1
                #self.failed = i_server
                return -1
            self.__servcon_obj[i_server].connect()

        if (i_exit_var == 0):
            if self.__d_config["project.name"] == "" or self.__d_config["test.name"] == "" or self.__d_config["tester.
                surname"] \
            == "" or self.__d_config["tester.forename"] == "":
                print "essential config-fields are empty like project.name, test.name or tester.name "
                return -1

            project_id = self.db_object.insert_project(str(self.__d_config["project.name"]), str(self.__d_config["
                project.desc"]))
            """ get project_id """

            tester_id = self.db_object.insert_tester(s_tester_surname = str(self.__d_config["tester.surname"]),
                s_tester_forename = \
                                                str(self.__d_config["tester.forename"]), s_tester_email = str(
                                                    self.__d_config["tester.email"]))
            """ get the tester_id """

            iteration_id = self.db_object.insert_test( str(self.__d_config["test.name"]), str(self.__d_config["test.
                desc"]), project_id, int(tester_id),self.__d_config["test.application"] )
            """ get test_id """
            if iteration_id == -1:
                #print "bad test"
                return -1


            self.ganglia_obj = GangliaThread(self.db_object,i_srbport,iteration_id,self.__d_config["measurement.
                poll_time"],\
                                            quantity = self.__d_config["measurement.quantity"],measurement_app = \
                                            self.__d_config["measurement.application"], i_verbose = self.__i_verbose)
            self.ganglia_obj.start()

        return 0


    def stop(self):
        """ stop the measurement """

        for i_server in range(int(self.__d_config["server.quantity"])):
```

```
            self.ganglia_obj.stop()
            if ( self.__servcon_obj != {} and self.__servcon_obj[i_server].connected == 0):
                #print "Hallo"

                self.__servcon_obj[i_server].stop_connection()



    def __get_srbport(self):
        """ parse for a srbport in the user folder + .srb/.MdasEnv """
        s_buf = ""
        srb_path = os.path.expanduser("~/.srb/.MdasEnv")
        try:
            file_desc = open(srb_path, 'r')
        except:
            print "cannot find any SRB-Port"
            return None
        s_buf = file_desc.readlines()
        for line in s_buf:
            #if x.find("srbPort"):
            srbport_begin = line.find("srbPort")
            if srbport_begin != -1 and (srbport_begin ==0 or line[0:srbport_begin].find("#") == -1):
                srb_port = line[(line.find("t")+1):line.find('#')]
                srb_port = srb_port.replace("""'""","""" """)
                srb_port = srb_port.replace('"',' ')
                srb_port = srb_port.strip()
                if self.__i_verbose:
                    print "found the following srb-Port: %s" % (srb_port)

        file_desc.close()
        return srb_port


class Crunapp(threading.Thread):
    """ start the application in a new Thread """
    def __init__(self, app,i_verbose = 0):
        """ Constructor to setup the Thread and the application """
        threading.Thread.__init__(self)
        #self.setDaemon(True)
        #self._set_daemon()
        self.__application = app
        self.__s_pid = ""
        self.__i_verbose = i_verbose

    def run(self):
        """ start the application in a new Thread """
        self.__s_pid = commands.getoutput("ps u -C '"+self.__application+"' | grep $USER | awk '{print $2}'");

        if self.__s_pid == '':
            #print self.isDaemon()
            try:
                if self.__i_verbose:
                    print "application will be started"
                print commands.getoutput(self.__application)
            except:
                print "Cannot start the application"
        else:

            print "programme already running"
        if self.__i_verbose:
            print "Crunapp is dead"

    def stop(self):
        self.__s_pid = commands.getoutput("ps u -C '"+self.__application+"' | grep $USER | awk '{print $2}'");
        if self.__i_verbose:
            print "PID of test application: ", self.__s_pid
        if self.__s_pid != '':
            print commands.getoutput("kill -9 %s" % self.__s_pid)
            if self.__i_verbose:
                print "Test application killed"
            self.__s_pid = ''
```

```
        else:
            if self.__i_verbose:
                print "No Test application killed"
```

B.5 console.py

```
#!/usr/bin/env python

""" console application to connect to other server and get information from the database"""

__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "18.10.2005"
__version__ = "0.1"
__revision__ = "1.0"


import sys
import getopt
import python_client
import re
import threading
import time

class CnoGUI:
    """ class for console work """

    def __init__(self):
        """ Constructor to get the options and declare the objects for the start function """
        try:
            self.__opts, self.__args = getopt.getopt(sys.argv[1:], 'hgi:t:p:m:d:s:e:c:f:v', ["help", "projectlist","
                iterationlist", "testlist", "srbserverlist", "measurementlist", "verbose", "project-id=", \
                                                                "test-id=","iteration-id=", "srbserver-id=", "
                                                                    database=","startday=","endday=", "
                                                                    config-file=", "config-dump=","h","gauge
                                                                    " ])
        except getopt.error:
            print "argument mistakes"
            self.__usage()
            sys.exit(-1)
        self._i_verbose = 0

        self._cparser = ""
        self._measure_obj = ""
        self.__d_startoptions = {}

    def start(self):
        """ parse the options and start the GUI or the command line programme """

        list_set = 0
        config_file = None
        database_file = None
        for s_option, s_argument in self.__opts:
            if (s_option.find('list') != -1):
                if list_set == 1:
                    print "ArguemnentError: just one list can be displayed"
                    self.__usage()
                    sys.exit(0)
                self.__d_startoptions[s_option] = 1
                list_set = 1
            elif s_option in ("-p", "--project-id"):
                self.__d_startoptions['project'] = s_argument
                reg_obj = re.match("([^0-9])|((\D)*(\d)+(\D)+)", s_argument)
                if reg_obj != None:
                    print "project-id needs to be a number"
                    sys.exit(0)
```

```
            elif s_option in ("-t", "--test-id"):
                self.__d_startoptions['test'] = s_argument
                reg_obj = re.match("([^0-9])|((\D)*(\d)+(\D)+)", s_argument)

                if reg_obj != None:
                    print "test-id needs to be a number"
                    sys.exit(0)
            elif s_option in ("-i", "--iteration-id"):
                self.__d_startoptions['iteration'] = s_argument
                reg_obj = re.match("([^0-9])|((\D)*(\d)+(\D)+)", s_argument)
                if reg_obj != None:
                    print "iteration-id needs to be a number"
                    sys.exit(0)
            elif s_option in ("-s", "--startday"):
                self.__d_startoptions['startday'] = s_argument
                var = re.match("[0-9]{4}\-[0-9]{2}\-[0-9]{2}", s_argument)
                if var == None:
                    print "bad date, use e.g. 2005-11-01"
                    sys.exit(0)
            elif s_option in ("-e", "--endday"):
                self.__d_startoptions['endday'] = s_argument
                var = re.match("[0-9]{4}\-[0-9]{2}\-[0-9]{2}", s_argument)
                if var == None:
                    print "bad date, use e.g. 2005-11-01"
                    sys.exit(0)
            elif s_option in ("-m", "--srbserver-id"):
                self.__d_startoptions['srbserver'] = s_argument
                reg_obj = re.match("([^0-9])|((\D)*(\d)+(\D)+)", s_argument)

                if reg_obj != None:
                    print "srbserver-id needs to be a number"
                    sys.exit(0)
            elif s_option in ("-c", "--config-file"):
                config_file = s_argument
            elif s_option in ("-d", "--datbase"):
                database_file = s_argument
            elif s_option in ("-f", "--config-dump"):
                cparser = python_client.Cconfigparser()
                cparser.write(s_argument)
                sys.exit(0)
            elif s_option in ("-v", "--verbose"):
                self._i_verbose = 1
            elif s_option in ("-h", "--help", "--h"):
                self.__usage()
                sys.exit(0)
            elif s_option in ("-g","--gauge"):
                self.__d_startoptions['measure'] = s_argument

            else:
                "print not the correct options"
                self.__usage()
                sys.exit(0)

        self._cparser = python_client.Cconfigparser(self._i_verbose)
        self._measure_obj = python_client.CMeasurement(database_file , self._i_verbose)
        self._measure_obj.set_config(self._cparser.load_config(config_file))
        if self.__d_startoptions == {}:
            self.__usage()

        else:

            self.__console__()

    def __console__(self):

        """ ***********************console application**************************** """

        """ *************project list**************************"""
        if self.__d_startoptions.has_key('--projectlist') == True:
            d_project_list = self._measure_obj.db_object.select_table("project","project_id, title, description");
            print "******* Project list ********"
```

```
            print "\n| project-id |        title         |    description                |"
            print " ------------------------------------------------------------------- "
            for project in d_project_list:
                print "| %10s | %20s | %30s |" % ( project['project_id'], project['title'], project['description'])

        elif self.__d_startoptions.has_key('--testlist') == True:
            self.__testlist()
        elif self.__d_startoptions.has_key('--iterationlist') == True:
            self.__iterationlist()
        elif self.__d_startoptions.has_key('--srbserverlist'):
            self.__srbserverlist()

        elif self.__d_startoptions.has_key('--measurementlist'):
            print "Measurements"
            self.__measurementlist()
        elif self.__d_startoptions.has_key('measure'):
            self.__measure()
        else:
            print "The programme needs at least a project-id or it is able to return a projectlist"
            self.__usage()
            sys.exit(0)


    def __srbserverlist(self):
        """ ******************** create a srbserver list ***************************"""

        if self.__d_startoptions.has_key('test') == True:

            self._measure_obj.db_object.db_cursor.execute(""" SELECT DISTINCT S.srbserver_id, H.hostname,
                                                H.ip_address, H.cpu_speed, H.memory, S.srb_port
                                                From srbserver AS S, host AS H
                                                INNER JOIN iteration AS I, iteration_srbserver AS I_S ON
                                                (I.test_id = """+self.__d_startoptions['test']+""")
                                                AND I_S.iteration_id = I.iteration_id
                                                AND I_S.srbserver_id = S.srbserver_id AND S.host_id = H.
                                                     host_id""")

            d_srbserver_list = self._measure_obj.db_object.db_cursor.fetchall()

        elif self.__d_startoptions.has_key('project') == True:
            self._measure_obj.db_object.db_cursor.execute(""" SELECT DISTINCT S.srbserver_id, H.hostname, H.ip_address
                 , H.cpu_speed, H.memory, S.srb_port
                                                 From srbserver AS S, host AS H
                                                 INNER JOIN test AS T, iteration AS I, iteration_srbserver AS
                                                      I_S, project AS P
                                                 where P.project_id = """+self.__d_startoptions['project']+"""
                                                 AND T.project_id = P.project_id AND T.test_id = I.test_id
                                                 AND I.iteration_id = I_S.iteration_id AND
                                                 I_S.srbserver_id = S.srbserver_id AND S.host_id = H.host_id""
                                                      ")

            d_srbserver_list = self._measure_obj.db_object.db_cursor.fetchall()

        else:
            self._measure_obj.db_object.db_cursor.execute(""" SELECT S.srbserver_id, H.hostname, H.ip_address, H.
                 cpu_speed,
                                                   H.memory, S.srb_port FROM
                                                   host AS H, srbserver AS S
                                                   WHERE H.host_id = S.host_id""")
            d_srbserver_list = self._measure_obj.db_object.db_cursor.fetchall()
        print "\nUsed srbserver:"
        print "\n| srbserver-id |         hostname         | srb-port | cpu-speed | memory  |"
        print " "+74*"-"
        for d_srbserver in d_srbserver_list:
            print "| %12s | %24s | %8s | %9s | %7s |" % ( d_srbserver['S.srbserver_id'], d_srbserver['H.hostname'],
                 d_srbserver['S.srb_port'], \
                                                d_srbserver['H.cpu_speed'], d_srbserver['H.memory'])
        print "\n"


    def __testlist(self):
```

```
        """ ****************** create a test list *************************"""

        if self.__d_startoptions.has_key('project') == True:
            if self.__d_startoptions.has_key('srbserver') == True:

                print "\n****** Testlist for project: %s and srbserver: %s" % (self.__d_startoptions["project"], self.
                    __d_startoptions["srbserver"])

                self._measure_obj.db_object.db_cursor.execute("""SELECT DISTINCT T.test_id, P.title, T.name, T.
                    description
                                                          FROM test AS T, project AS P
                                                         INNER JOIN iteration AS I, iteration_srbserver AS I_S
                                                         where I_S.srbserver_id = '%s' AND I_S.iteration_id = I.
                                                              iteration_id
                                                         AND T.project_id = P.project_id AND I.test_id = T.test_id
                                                              AND
                                                          P.project_id = '%s'""" \
                                                        % (self.__d_startoptions['srbserver'], self.
                                                            __d_startoptions['project']) )
            else:
                print "\n****** Testlist for project: %s " % (self.__d_startoptions["project"])

                self._measure_obj.db_object.db_cursor.execute(""" SELECT T.test_id, T.name, T.description, P.title
                                                          FROM test AS T, project AS P
                                                          WHERE T.project_id = '%s' AND T.project_id = P.project_id
                                                               """ \
                                                          % self.__d_startoptions['project'])


        elif self.__d_startoptions.has_key('srbserver') == True:
            print "\n****** Testlist for srbserver: %s " % (self.__d_startoptions["srbserver"])

            self._measure_obj.db_object.db_cursor.execute("""SELECT DISTINCT T.test_id, P.title, T.name, T.description
                                                   FROM test AS T, project AS P
                                                  INNER JOIN iteration AS I, iteration_srbserver AS I_S
                                                  where I_S.srbserver_id = '%s' AND I.iteration_id = I_S.
                                                       iteration_id
                                                  AND T.project_id = P.project_id AND I.test_id = T.test_id """
                                                       \
                                                 % (self.__d_startoptions['srbserver']) )


        else:
            print "Error: A project-id or an srbserver_id is needed for the testlist\nTry console.py --help\n"
            sys.exit(0)

        #### print testlist
        d_test_list = self._measure_obj.db_object.db_cursor.fetchall()
        print "\n| test-id |     title       | project-title  |       description         |"
        print " "+73*"-"
        for d_test in d_test_list:

            if d_test != None:
                print "| %7s | %15s | %15s | %25s |" % (d_test['T.test_id'], d_test['T.name'], d_test['P.title'],
                    d_test['T.description'])
        print "\n"


    def __iterationlist(self):
        """ create a iteration list (reused tests) """
        if self.__d_startoptions.has_key('test') == True:
            if self.__d_startoptions.has_key('srbserver') == True:

                self._measure_obj.db_object.db_cursor.execute("""SELECT I.*, T.name FROM iteration AS I, test AS T
                                                          INNER JOIN iteration_srbserver AS I_S WHERE
                                                          T.test_id = '%s' AND I.test_id = T.test_id AND
                                                   I_S.srbserver_id = '%s' AND I_S.iteration_id = I.iteration_id
                                                   """ % (self.__d_startoptions["test"], self.__d_startoptions["
                                                        srbserver"]) )
            else:
                self._measure_obj.db_object.db_cursor.execute("""SELECT I.*, T.name FROM iteration AS I, test AS T
```

```
                                                    WHERE T.test_id = '%s' AND I.test_id = T.test_id""" % self.
                                                        __d_startoptions["test"] )
            d_measurements = self._measure_obj.db_object.db_cursor.fetchall()
            #print d_measurements
            print "\n| iteration-id |   test-title    |           time          |"
            print " "+60*"-"
            for d_test in d_measurements:

                if d_test != None:
                    print "| %12s | %15s | %25s |" % (d_test['I.iteration_id'], d_test['T.name'], d_test['I.time'])
            print "\n"
        else:
            self.__usage()

    def __measurementlist(self):
        """ ********************** create a measurement list ******************************"""

        if self.__d_startoptions.has_key('iteration') == True:

            #******************** measurementlist with only a srbserver-id **********************
            self._measure_obj.db_object.db_cursor.execute("""SELECT I_S.srbserver_id,M.* FROM measurement AS M,
                iteration_srbserver AS I_S
                                                     INNER JOIN iteration AS I
                                                    WHERE I.iteration_id = '%s' AND I.iteration_id = I_S.
                                                        iteration_id
                                                       AND I_S.is_id = M.is_id ORDER BY I_S.srbserver_id """ \
                                                    % self.__d_startoptions["iteration"] )
            d_measurements = self._measure_obj.db_object.db_cursor.fetchall()
                #print d_measurements

        #*************** measurementlist with only a test-id ***********************
        elif self.__d_startoptions.has_key('srbserver') == True:
            self._measure_obj.db_object.db_cursor.execute("""SELECT M.* FROM measurement AS M INNER JOIN
                iteration_srbserver AS I_S
                                                    WHERE I_S.srbserver_id = '%s' AND I_S.is_id = M.is_id""" %
                                                        self.__d_startoptions["srbserver"] )
            d_measurements = self._measure_obj.db_object.db_cursor.fetchall()


        #*************** with start-day and endday ********************

        elif self.__d_startoptions.has_key('startday') == True and self.__d_startoptions.has_key('endday'):
            self._measure_obj.db_object.db_cursor.execute("""SELECT M.* FROM measurement AS M where M.time BETWEEN '%s
                ' and '%s'
                                                    """ %(self.__d_startoptions['startday'], self.
                                                        __d_startoptions['endday']) )
            d_measurements = self._measure_obj.db_object.db_cursor.fetchall()
            #print d_measurements
        else:
            print "ERROR: need at least a iteration-id or/and a srbserver-id "
            self.__usage()
            sys.exit(0)

        for d_measurement in d_measurements:
            print ""
            for s_key, s_value in d_measurement.items():
                point_delete = s_key.find(".")
                print "%20s : %s" % (s_key[point_delete+1:], s_value)
            print "\n-----------------------------------------------"

    def __measure(self):
        """ start a measurement in the console """
        i_measurement_return = self._measure_obj.run_measurements()

        ## a server connection failed, stop all other connections ##
        if (i_measurement_return == -1):
            print "no connection reached"
            self._measure_obj.stop_connections()
        ## all server connected
        else:
            ## set all measurements to zero (for the dynamic table and the graph
```

```
            ## ask for graphic support
            config = self._measure_obj.get_config()
            measurement_counter = 0
            try:
                while not self._measure_obj.ganglia_obj._stopevent.isSet():
                    time.sleep(1)
                    l_measurement = self._measure_obj.db_object.get_measurements()
                    if l_measurement != []:
                        print l_measurement
                        measurement_counter += 1
                        for d_measurement in l_measurement:
                            print ""
                            for s_key, s_value in d_measurement.items():

                                print "%20s : %s" % (s_key, s_value)
                            print "\n-----------------------------------------------"
            except KeyboardInterrupt:
                #self._measure_obj.servcon_obj.ststop_connections()
                print "closing the connection and the Threads PLEASE WAIT!!"
                self._measure_obj.stop()
                return 0
            self._measure_obj.stop()
            #self._measure_obj.stop_connections()

    def __usage(self):
        """ A little help for the user """
        print """
Usage: pyhon_server.py

        --projectlist

        --testlist
        (needs project-id and/or srbserver-id)

        --srbserverlist
        (can be used with project-id and/or test-id)

        --iterationlist
        (needs test-id can be used with srbserver-id)

        --measurementlist
        (needs iteration or/and srbserver-id or
        (-start and -endday))

        Use the ids to get the lists
        -p <project-id>      --project-id=
        -t <test-id>         --test-id=
        -i <iteration-id>    --iteration-id=
        -m <srbserver-id>    --srbserver-id=

        See the measurements between two days
        -s <YYYY-MM-DD>      --startday=
        -e <YYYY-MM-DD>      --endday=

        Set a database (standard srb.db)
        -d <database>        --database=

        Set a config file
        -c <config-file>     --config-file=

        Write a config-file
        -f <config-file>     --config-dump=

        Start a measurement
        -g                   --gauge

        See more information during the use of the application
        -v                   --verbose
        See this again
        -h --help --h
```

```
        """

def main():
    """ main function """
    try:
        CnoGUI().start()
    except KeyboardInterrupt:
        print "Hallo"


if __name__ == '__main__':
    sys.exit(main())
```

B.6 gui2.py

```
#!/usr/bin/env python

"""
* Python gui to connect to a server application
*
* sip04ck
*
"""
__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "17.10.2005"
__version__ = "0.1"
__revision__ = "1.0"

import gui2_ui
import Tkinter
import sys
import getopt
import python_client
import re
import threading
import time

class CPrintinGUI:
    """ print console output into gui console """

    def __init__(self, func):
        """ constructor to initiliaze variables """
        self.out = 1
        self.func = func

    def write(self, s_text):
        """ overwrite write function of the stdout """
        self.func(s_text)

    def stop(self):
        """ stop writing into the console """
        sys.stdout = sys.__stdout__

class Cguistart:
    def __init__(self):
        """ Constructor to get the options and declare the objects for the start function """
        try:
            self.__opts, self.__args = getopt.getopt(sys.argv[1:], 'hd:c:f:v', ["help", "database=","config-file=", "
                config-dump=","h",])
        except getopt.error:
            print "argument mistakes"
            self.usage()
            sys.exit(-1)
        self._i_verbose = 0
```

```
        self._cparser = ""
        self._measure_obj = ""
        self.__d_startoptions = {}
        self.threads_connected = threading.Event()

    def start(self):
        """ get the information from the commandline and start the gui"""
        config_file = None
        database_file = None
        for s_option, s_argument in self.__opts:
            if s_option in ("-c", "--config-file"):
                config_file = s_argument
            elif s_option in ("-d", "--datbase"):
                database_file = s_argument
            elif s_option in ("-f", "--config-dump"):
                cparser = python_client.Cconfigparser()
                cparser.write(s_argument)
                sys.exit(0)
            elif s_option == "-v":
                self._i_verbose = 1
            elif s_option in ("-h", "--help", "--h"):
                self.usage()
                sys.exit(0)
            else:
                print "not the correct options"
                self.usage()
                sys.exit(0)

        self._cparser = python_client.Cconfigparser(self._i_verbose)
        self._measure_obj = python_client.CMeasurement(database_file, self._i_verbose)
        self._measure_obj.set_config(self._cparser.load_config(config_file))

        self.gui()

    def usage(self):
        """ A little help for the user """
        print """
Usage: -d <database-file>
       -f <config-dump>
       -c <config-file>
       -h
       """



    def gui(self):
        """start the GUI """

        demo = gui2_ui.Dialog(self)
        demo.protocol('WM_DELETE_WINDOW', demo._gui_quit)
        """ print the console output in the GUI """
        var = CPrintinGUI(demo._change_text_in_console)
        #sys.stdout = var
        #sys.stderr = var

        """ initialise the main window"""
        #root.title('SRB Benchmark')
        demo.geometry("700x700+300+80")

        try:
            demo.mainloop()
           # print "end"

        except KeyboardInterrupt:

            try:
                self._measure_obj.stop()
                self._measure_obj.db_object.db_cursor.close()
                self._measure_obj.db_object.db_con.close()
            except:
```

```
                print "SQLite db already closed"
            print "quit now, please use the x or the exit button in the menu"

def main():
    """ main function """

    Cguistart().start()


if __name__ == '__main__':
    sys.exit(main())
```

B.7 gui2_ui.py

```
#!/usr/bin/env python

"""
* Python second main_gui
*
* sip04ck
*
"""
__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "17.10.2005"
__version__ = "0.1"
__revision__ = "1.0"

import Tkinter
import time
import thread
import tkFileDialog
import tkMessageBox
import tkSimpleDialog
import shutil
import myplot
import threading
import sys

class AutoScrollbar(Tkinter.Scrollbar):
    """  a scrollbar that hides itself if it's not needed.  only
     works if you use the grid geometry manager.
    """
    def set(self, f_low, f_high):
        """ overwrite the set function of the Scrollbar class """
        if float(f_low) <= 0.0 and float(f_high) >= 1.0:
            # grid_remove is currently missing from Tkinter!
            self.tk.call("grid", "remove", self)
        else:
            self.grid()
        Tkinter.Scrollbar.set(self, f_low, f_high)

class Mainframe(Tkinter.Frame):
    """****************  MAIN FRAME *******************************"""
    def __init__(self,parent,app = None):
        """constructor to initialise the frame (autoscrollbars and canvas) """


        Tkinter.Frame.__init__(self, parent)
        self._parent = parent

        Tkinter.Frame.__init__(self,self._parent)
        self.configure(
             self._parent._root,
        )
```

```
        self.grid_columnconfigure(0, weight = 1, minsize = 10, pad = 0)
        self.grid_rowconfigure(0, weight = 1, minsize = 10, pad = 0)
        self.__vscrollbar = AutoScrollbar(self)
        self.__vscrollbar.grid(
            in_      = self,
            column   = 1,
            row      = 0,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'ns'
        )
        self.__hscrollbar = AutoScrollbar(self, orient="horizontal")
        self.__hscrollbar.grid(row=1, column=0, sticky="ew")
        self.grid_columnconfigure(1, weight = 0, minsize = 10, pad = 0)

        self.grid_rowconfigure(1, weight=0)
        self._canvas = Tkinter.Canvas(
            self,
            yscrollcommand=self.__vscrollbar.set,
            xscrollcommand=self.__hscrollbar.set,

        )
        self._frame = Tkinter.Frame(self._canvas, )


    def config(self):
        """destroy the frame and initialise it new """
        self._frame.destroy()
        self._canvas.destroy()
        self._canvas = Tkinter.Canvas(
            self,
            yscrollcommand=self.__vscrollbar.set,
            xscrollcommand=self.__hscrollbar.set,

        )
        self._canvas.grid(
            column   = 0,
            row      = 0,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nwes'
            )

        self._canvas.grid_columnconfigure(0, weight=1)
        self._canvas.grid_rowconfigure(0, weight=1)


        self.__vscrollbar.config(command = self._canvas.yview)
        self.__hscrollbar.config(command = self._canvas.xview)

        self._frame = Tkinter.Frame(self._canvas, )
        self._frame.grid(
            column = 0,
            row    = 0,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
```

```
            sticky = 'nwes'
            )

    def _canvas_reload(self):
        """refresh the canvas so it is able to make the frame scrollable """

        self._canvas.create_window(0, 0, anchor = "nw", window = self._frame)
        self._frame.update_idletasks()
        self._canvas.config(scrollregion = self._canvas.bbox("all"))

class graph_choice:
    """ The main frame with 4 different listboxes to choose the graphs"""
    def __init__(self,parent,app):
        """ constructor """
        self._parent = parent

        if app != None:

            self.__app = app
        self.__project = None
        self.__srbserver = None
        self.__test = None

    def config(self):
        """ set the frame for the tables to get the measurements of the database """
        self._parent._main_frame.config()
        self._frame = self._parent._main_frame._frame
        self._parent._active = "main"

        label_title = Tkinter.Label(self._frame, text = "SRB database information")
        label_title.grid(
            in_     = self._frame,
            column  = 0,
            row     = 0,
            columnspan = '5',
            ipadx = '8',
            ipady = '8',
            padx = '5',
            pady = '5',
            rowspan = '1',
            sticky = 'nwes'
                            )
        button_project = Tkinter.Button(self._frame, text = "Projects", command = self._project_table, state = "
            disabled",)
        button_project.grid(
            in_     = self._frame,
            column  = 1,
            row     = 1,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '5',
            pady = '5',
            rowspan = '1',
            sticky = ''
        )

        button_srbserver = Tkinter.Button(self._frame, text = "srbservers", command = self._srbserver_table, state = "
            disabled")
        button_srbserver.grid(
            in_     = self._frame,
            column  = 4,
            row     = 1,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '1',
            pady = '1',
            rowspan = '1',
            sticky = ''
```

```
        )

        self.__select_box_project = Tkinter.Listbox(self._frame, selectmode="single", bg = "white")
        self.__select_box_project.grid(
            in_      = self._frame,
            column   = 1,
            row      = 2,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nw'
        )
        self.__select_box_srbserver = Tkinter.Listbox(self._frame, selectmode="single", width = 30, bg = "white")
        self.__select_box_srbserver.grid(
            in_      = self._frame,
            column   = 4,
            row      = 2,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nw'
        )

        self.__app._measure_obj.db_object.lock_it()

        try:
            self.__app._measure_obj.db_object.db_cursor.execute("SELECT title FROM project ORDER BY title")
        except:
            print "Bad database file"
            sys.exit(-1)
            self.__app._measure_obj.db_object.rlock()
        project_table = self.__app._measure_obj.db_object.db_cursor.fetchall()
        self.__app._measure_obj.db_object.rlock()

        if len(project_table) != 0:
            button_project.config(state = "active")

        for project in project_table:
           self.__select_box_project.insert("end", project[0])

        self.__app._measure_obj.db_object.lock_it()
        self.__app._measure_obj.db_object.db_cursor.execute("""SELECT host.hostname, srbserver.srb_port
                FROM srbserver, host WHERE srbserver.host_id = host.host_id""")
        srbserver_table = self.__app._measure_obj.db_object.db_cursor.fetchall()
        self.__app._measure_obj.db_object.rlock()
        #print srbserver_table
        if len(srbserver_table) != 0:
            srbserver_table.sort()
            button_srbserver.config(state = "active")
        for x in srbserver_table:
            self.__select_box_srbserver.insert("end", str(x["srbserver.srb_port"])+": "+x["host.hostname"])

        select_scrollbar_project = Tkinter.Scrollbar(self._frame)
        self.__select_box_project.configure(
            yscrollcommand = select_scrollbar_project.set

        )

        select_scrollbar_project.configure(
            command = self.__select_box_project.yview,

                                           )
        select_scrollbar_project.grid(
            in_      = self._frame,
            column   = 2,
```

```
            row      = 2,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nws',

        )
        select_scrollbar_srbserver = Tkinter.Scrollbar(self._frame)
        select_scrollbar_srbserver_h = AutoScrollbar(self._frame, orient="horizontal")

        self.__select_box_srbserver.configure(
            yscrollcommand = select_scrollbar_srbserver.set,
            xscrollcommand = select_scrollbar_srbserver_h.set
        )

        select_scrollbar_srbserver.configure(
            command = self.__select_box_srbserver.yview,
        )
        select_scrollbar_srbserver_h.configure(
            command = self.__select_box_srbserver.xview
        )

        select_scrollbar_srbserver.grid(
            in_      = self._frame,
            column   = 5,
            row      = 2,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nws',

        )
        select_scrollbar_srbserver_h.grid(
            in_      = self._frame,
            column   = 4,
            row      = 3,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nwes',

        )

        self.__button_set_project = Tkinter.Button(
            self._frame,
            text= "set Project",
            command = self.__get_project
        )
        self.__button_set_project.grid(
            in_      = self._frame,
            column   = 1,
            row      = 4,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '5',
            rowspan = '1',
            sticky = 'n',
        )

        self.__button_set_srbserver = Tkinter.Button(
```

```
            self._frame,
            text = "set Srbserver",
            command = self.__get_srbserver
            )

        self.__button_set_srbserver.grid(
            in_     = self._frame,
            column  = 4,
            row     = 4,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '5',
            rowspan = '1',
            sticky = 'n',
        )
        if self.__srbserver or self.__project:
            if self.__srbserver:
                self.__button_set_srbserver.configure(text = "unset srbserver")
            if self.__project:
                self.__button_set_project.configure(text = "unset Project")
                self.__set_test()

        self._frame.grid_columnconfigure(0, weight = 0, minsize = 5, pad = 0)
        self._frame.grid_rowconfigure(0, weight = 0, minsize = 5, pad = 0)
        self._frame.grid_columnconfigure(1, weight = 0, minsize = 10, pad = 0)
        self._frame.grid_rowconfigure(1, weight = 0, minsize = 10, pad = 0)
        self._frame.grid_columnconfigure(2, weight = 0, minsize = 10, pad = 0)
        self._frame.grid_rowconfigure(2, weight = 0, minsize = 10, pad = 0)
        self._frame.grid_columnconfigure(3, weight = 1, minsize = 10, pad = 0)
        self._frame.grid_rowconfigure(3, weight = 0, minsize = 0, pad = 0)
        self._frame.grid_columnconfigure(4, weight = 0, minsize = 10, pad = 0)
        self._frame.grid_rowconfigure(4, weight = 0, minsize = 10, pad = 0)

        self._parent._main_frame._canvas_reload()

    def __set_test(self):
        """ if a srbserver and a project or just a project has been chosen
            in the main frame the test table will be shown to choose a test
            and the iteration table will be shown
        """
        test_label = Tkinter.Button(self._frame, text = "Tests", command = self._test_table)

        self.__select_box_test = Tkinter.Listbox(self._frame, selectmode = "single", bg = "white")

        select_scrollbar_test = Tkinter.Scrollbar(self._frame)

        self.__select_box_test.configure(
            yscrollcommand = select_scrollbar_test.set
        )

        select_scrollbar_test.configure(

            command = self.__select_box_test.yview,
        )

        if self.__project != None:
            self.__app._measure_obj.db_object.lock_it()
            if self.__srbserver != None:

                self.__app._measure_obj.db_object.db_cursor.execute("""SELECT DISTINCT name
                        FROM test,iteration INNER JOIN iteration_srbserver where iteration_srbserver.srbserver_id =
                        (Select srbserver.srbserver_id from srbserver, host
                                where host.hostname = '%s' AND srb_port = %d and host.host_id = srbserver.host_id)
                        AND test.project_id = (Select project.project_id from project where project.title = '%s')
                        AND test.test_id = iteration.test_id AND iteration.iteration_id = iteration_srbserver.
                              iteration_id AND
                        test.failed_flag = 0""" % (str(self.__srbserver),int(self.__srb_port),str(self.__project)))

                tests = self.__app._measure_obj.db_object.db_cursor.fetchall()
```

```
405
                if tests == []:
                    self.__srbserver = None
                    print "no test with this project and the selected srbserver"
                    self.config()
410
            else:
                self.__app._measure_obj.db_object.db_cursor.execute(""" SELECT test.name,test_id
                        FROM test INNER JOIN project WHERE project.title = '%s'
                        AND test.project_id = project.project_id AND test.failed_flag = 0""" % self.__project )
415
                tests = self.__app._measure_obj.db_object.db_cursor.fetchall()
                self.__select_box_srbserver.delete(0,"end")
                self.__app._measure_obj.db_object.db_cursor.execute(""" SELECT DISTINCT H.hostname, S.srb_port
                        FROM host AS H, srbserver AS S
420                     INNER JOIN project as P ON (P.project_id = T.project_id) AND P.title = '%s'
                        INNER JOIN iteration_srbserver AS I_S ON (I_S.srbserver_id = S.srbserver_id)
                        INNER JOIN iteration AS I ON (I.iteration_id = I_S.iteration_id)
                        INNER JOIN test AS T ON (I.test_id = T.test_id)
                        AND H.host_id = S.host_id""" % self.__project)
425             new_srbservers = self.__app._measure_obj.db_object.db_cursor.fetchall()
                #print new_srbservers
                new_srbservers.sort()
                for srbserver in new_srbservers:
                    self.__select_box_srbserver.insert("end", str(srbserver["S.srb_port"])+": "+str(srbserver["H.
                        hostname"]))
430         self.__app._measure_obj.db_object.rlock()
            for test in tests:
                self.__select_box_test.insert("end", test[0])
            test_label.grid(
                in_     = self._frame,
435             column  = 1,
                row     = 5,
                columnspan = '1',
                ipadx = '0',
                ipady = '0',
440             padx = '0',
                pady = '5',
                rowspan = '1',
                sticky = '',
            )
445         self.__select_box_test.grid(
                in_     = self._frame,
                column  = 1,
                row     = 6,
                columnspan = '1',
450             ipadx = '0',
                ipady = '0',
                padx = '0',
                pady = '5',
                rowspan = '1',
455             sticky = 'n',
            )
            select_scrollbar_test.grid(
                in_     = self._frame,
                column  = 2,
                row     = 6,
460             columnspan = '1',
                ipadx = '0',
                ipady = '0',
                padx = '0',
465             pady = '5',
                rowspan = '1',
                sticky = 'nws',
            )

470         button_set_test = Tkinter.Button(
            self._frame,
            text = "set Test",
            command = self.__set_test_command
```

```
            )

            button_set_test.grid(
                column  = 1,
            row     = 7,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '5',
            rowspan = '1',
            sticky = '',
                )
            if self.__test != None:
                self.__set_test_command()


        elif self.__srbserver != None:
            self.__select_box_project.delete(0,"end")
            self.__app._measure_obj.db_object.lock_it()
            self.__app._measure_obj.db_object.db_cursor.execute(""" SELECT DISTINCT P.title FROM project AS P
                INNER JOIN srbserver as S, host AS H ON (S.srb_port = '%s') AND H.host_id = S.host_id AND H.hostname =
                      '%s'
                INNER JOIN iteration_srbserver AS I_S, iteration ON (I_S.srbserver_id = S.srbserver_id) AND
                (iteration.test_id = T.test_id) AND (iteration.iteration_id = I_S.iteration_id)
                INNER JOIN test AS T ON (P.project_id = T.project_id) ORDER BY P.title""" % (self.__srb_port,self.
                    __srbserver))
            d_projects = self.__app._measure_obj.db_object.db_cursor.fetchall()
            self.__app._measure_obj.db_object.rlock()
            for project in d_projects:
                self.__select_box_project.insert("end", project['P.title'])

        self._parent._main_frame._canvas_reload()

    def _project_table(self):
        """ show all projects in a table """
        self.__app._measure_obj.db_object.lock_it()
        self.__app._measure_obj.db_object.db_cursor.execute("""SELECT * FROM project""")
        d_project = self.__app._measure_obj.db_object.db_cursor.fetchall()
        self.__app._measure_obj.db_object.rlock()
        self._create_table(d_project,"Projectlist")

    def _test_table(self):
        """ show all tests in table """
        self.__app._measure_obj.db_object.lock_it()
        self.__app._measure_obj.db_object.db_cursor.execute("""SELECT T.failed_flag, T.test_id, T.name,
                T.description, TER.forename, TER.surname,TER.email FROM test AS T
                INNER JOIN project AS P ON (T.project_id = P.project_id) AND P.title = '%s'
                INNER JOIN tester AS TER ON (TER.tester_id = T.tester_id)""" % self.__project)
        d_tests = self.__app._measure_obj.db_object.db_cursor.fetchall()
        self.__app._measure_obj.db_object.rlock()
        self._create_table(d_tests,"Testlist")

    def _srbserver_table(self):
        """ show all srbservers in a table """
        self.__app._measure_obj.db_object.lock_it()
        self.__app._measure_obj.db_object.db_cursor.execute("""SELECT S.srbserver_id,
                H.hostname, S.srb_port, H.cpu_speed, H.memory, H.ip_address FROM host AS H, srbserver AS S
                WHERE H.host_id = S.host_id""")
        d_srbservers = self.__app._measure_obj.db_object.db_cursor.fetchall()
        self.__app._measure_obj.db_object.rlock()
        self._create_table(d_srbservers,"SRBserverlist")


    def _create_table(self, d_measurements, heading, append = 0):
        """ create a table in the main-frame for project, test and srbserver """
        self._parent._active = "create_table"
        d_meas_labels = {}
        self._parent._main_frame.config()
        self._frame = self._parent._main_frame._frame
        self._i_count = 0
```

```
        i_count = 0
        meas_heading = Tkinter.Label(self._frame, text = heading)
        meas_heading.grid(
            column = i_count,
            row    = 0,
            columnspan = len(d_measurements[0]),
            ipadx = '0',
            ipady = '0',
            padx = '5',
            pady = '5',
            rowspan = '1',
            sticky = 'nwes'
        )

        for headings in d_measurements[0].keys():
            point = headings.find(".")
            if point != -1:
                headings = headings[point+1:]
            d_meas_labels[headings] = Tkinter.Label(self._frame, text = headings, relief = "groove")
            d_meas_labels[headings].grid(
                column = i_count,
                row    = 1,
                columnspan = '1',
                ipadx = '0',
                ipady = '0',
                padx = '0',
                pady = '0',
                rowspan = '1',
                sticky = 'nwes'
                )
            i_count += 1


        i_count = 0
        column_begin = 0

        self._d_meas_entries = []
        self._failedVar = []

        for measurement in d_measurements:
            i_count2 = 0
            #print i_count
            self._d_meas_entries.append([])
            for content in measurement:

                if heading == "Testlist" and i_count2 == 0:
                    self._failedVar.append({})
                    self._failedVar[i_count]['failed_flag'] = Tkinter.IntVar()
                    self._failedVar[i_count]['failed_flag'].set(content)
                    self._d_meas_entries[i_count].append(Tkinter.Checkbutton(self._frame, variable = \
                         self._failedVar[i_count]['failed_flag'], onvalue = 1, offvalue = 0))

                    #d_meas_entries[i_count][i_count2].
                elif heading == "Testlist" and i_count2 == 1:
                    self._failedVar[i_count]['tm_id'] = Tkinter.StringVar()
                    self._failedVar[i_count]['tm_id'].set(content)
                    self._d_meas_entries[i_count].append(Tkinter.Label(self._frame, \
                        text = self._failedVar[i_count]['tm_id'].get(), relief = "ridge"))
                    self._failedVar[i_count]['tm_id'].set(content)
                else:
                   # print i_count
                    self._d_meas_entries[i_count].append( \
                                Tkinter.Label(self._frame, text = content, relief = "ridge"))

                self._d_meas_entries[i_count][i_count2].grid(
                    column = i_count2,
                    row    = i_count+2,
                    columnspan = '1',
                    ipadx = '0',
                    ipady = '0',
                    padx = '0',
```

```
                    pady = '0',
                    rowspan = '1',
                    sticky = 'nwes'
                )
                i_count2 += 1
            i_count += 1
        if heading == "Testlist":
            set_test_button = Tkinter.Button(
                self._frame,
                text = "Set Flags",
                command = self._set_test_flag
                                        )
            set_test_button.grid(
                column = 0,
                row    = i_count+2,
                columnspan = '1',
                ipadx = '0',
                ipady = '0',
                padx = '0',
                pady = '0',
                rowspan = '1',
                sticky = ''
                                )
        #d_meas_entries = {}
        self._parent._main_frame._canvas_reload()


    def __get_project(self):
        """ set a project in the main view """
        if self.__project != None:
            self.__button_set_project.configure(text = "set Project")
            self.__project = None
            self.config()

        elif self.__select_box_project.curselection() != ():
            self.__project = self.__select_box_project.get("active")
            self.__button_set_project.configure(text = "unset Project")
            self.__set_test()


    def __get_srbserver(self):
        """ set a srbserver in the main view"""
        if self.__srbserver != None:
            self.__button_set_srbserver.configure(text = "set srbserver")
            self.__srbserver = None
            self.__srb_port = None
            if self.__project == None:
                self.config()
            else:
                self.config()
                self.__set_test()

        elif self.__select_box_srbserver.curselection() != ():
            self.__srbserver = self.__select_box_srbserver.get("active")
            srb_var = self.__srbserver.find(":")
            self.__srb_port = self.__srbserver[:srb_var]
            self.__srbserver = self.__srbserver[srb_var+2:]
            self.__button_set_srbserver.configure(text = "unset srbserver")
            self.config()
            self.__set_test()

    def __set_test_command(self):
        """ this function sets the test in the main view and start the iteration table """
        if self.__select_box_test.curselection() != ():


            self.__test = self.__select_box_test.get("active")


            self.__select_box_times = Tkinter.Listbox(self._frame, selectmode = "extended", bg = "white")
```

```
685             select_scrollbar_times = Tkinter.Scrollbar(self._frame)
690             self.__select_box_times.configure(
                    yscrollcommand = select_scrollbar_times.set
                )
                select_scrollbar_times.configure(
695
                    command = self.__select_box_times.yview,
                )
                self.__app._measure_obj.db_object.lock_it()
700             if self.__srbserver != None and self.__srb_port != None :
                    self.__app._measure_obj.db_object.db_cursor.execute("""SELECT DISTINCT iteration.time FROM iteration
                          INNER JOIN iteration_srbserver AS I_S
                          ON I_S.srbserver_id = (SELECT srbserver.srbserver_id
705                             FROM srbserver,host WHERE host.host_id = srbserver.host_id AND
                          host.hostname = '%s' AND srbserver.srb_port = '%s')
                          AND iteration.iteration_id = I_S.iteration_id
                          AND iteration.test_id = (SELECT test.test_id FROM test INNER JOIN
                          project WHERE project.title = '%s' AND project.project_id = test.project_id AND test.name = '%s
                               ')""" \
710                         % (self.__srbserver, self.__srb_port, self.__project, self.__test))
                else:
                    self.__app._measure_obj.db_object.db_cursor.execute("""SELECT DISTINCT iteration.time FROM iteration
715                          INNER JOIN iteration_srbserver AS I_S
                          WHERE iteration.test_id = (SELECT test.test_id FROM test INNER JOIN
                          project WHERE project.title = '%s' AND project.project_id = test.project_id AND test.name = '%
                               s')
                          AND iteration.iteration_id = I_S.iteration_id
                          """ % (self.__project, self.__test))
720
                d_times = self.__app._measure_obj.db_object.db_cursor.fetchall()

                self.__app._measure_obj.db_object.rlock()
                for time_stamp in d_times:
725                 self.__select_box_times.insert("end", time_stamp[0])


                self.__select_box_times.grid(
                    in_      = self._frame,
730                 column   = 4,
                    row      = 6,
                    columnspan = '1',
                    ipadx = '0',
                    ipady = '0',
735                 padx = '0',
                    pady = '5',
                    rowspan = '1',
                    sticky = 'nwe',
                )
740             select_scrollbar_times.grid(
                    in_      = self._frame,
                    column   = 5,
                    row      = 6,
                    columnspan = '1',
745                 ipadx = '0',
                    ipady = '0',
                    padx = '0',
                    pady = '5',
                    rowspan = '1',
750                 sticky = 'nws',
                )
                button_get_measurement = Tkinter.Button(
                    self._frame,
```

```
                text = "get Measurements",
                command = self.__get_measurements
                )
            button_get_measurement.grid(
                in_     = self._frame,
                column  = 6,
                row     = 6,
                columnspan = '1',
                ipadx = '0',
                ipady = '0',
                padx = '0',
                pady = '5',
                rowspan = '1',
                sticky = '',
            )

    def __get_measurements(self, view = "graph"):
        """ get the measurements and create a measurement table """
        if self.__select_box_times.curselection() != ():
            self.__times = []
            selection = self.__select_box_times.curselection()
            s_times = ""
            for x in selection:
                i_time = self.__select_box_times.get(x)
                self.__times.append(i_time)
                s_times += " I.time = '%s' OR" % i_time

            self.__app._measure_obj.db_object.lock_it()

            if self.__srbserver != None and self.__srb_port != None :

                select_string = """ SELECT DISTINCT M.* from measurement AS M
                                    INNER JOIN project AS P, test AS T on (P.project_id = T.project_id) AND P.title =
                                        '%s'
                                    INNER JOIN iteration AS I, iteration_srbserver as I_S ON (T.test_id = I.test_id)
                                    AND I_S.iteration_id = I.iteration_id AND ( %s ) AND T.name = '%s'
                                    AND I_S.srbserver_id = (SELECT srbserver.srbserver_id
                                        FROM srbserver,host WHERE host.host_id = srbserver.host_id AND
                                        host.hostname = '%s' AND srbserver.srb_port = '%s')
                                    AND M.is_id = I_S.is_id
                                    """ %(self.__project,s_times[:-2], self.__test, self.__srbserver, self.__srb_port)
                print select_string
                self.__app._measure_obj.db_object.db_cursor.execute(select_string)

            else:

                self.__app._measure_obj.db_object.db_cursor.execute(""" SELECT DISTINCT M.* from measurement AS M
                        INNER JOIN project AS P, test AS T on (P.project_id = T.project_id) AND P.title = '%s'
                        INNER JOIN iteration AS I, iteration_srbserver as I_S ON (T.test_id = I.test_id)
                        AND I_S.iteration_id = I.iteration_id AND ( %s )
                        AND M.is_id = I_S.is_id
                        """ %(self.__project,s_times[:-2]))

            d_measurements = self.__app._measure_obj.db_object.db_cursor.fetchall()
            print d_measurements
            self.__app._measure_obj.db_object.rlock()
            self.list = d_measurements
            myplot.Graph(self.__app._measure_obj.db_object).config(self.list)



    def _set_test_flag(self):
        """ set the flag if a test shouldn't be seen in the table """
        #print self._d_meas_entries[0]
        i_count = 0
        self.__app._measure_obj.db_object.db_cursor.execute("""Select failed_flag from test""")
        old_flags = self.__app._measure_obj.db_object.db_cursor.fetchall()
        for failed in self._failedVar:
            new_flag = failed['failed_flag'].get()
            self.__app._measure_obj.db_object.db_cursor.execute("""SELECT count(failed_flag) from test where
                  failed_flag = 0""")
```

```
            test = self.__app._measure_obj.db_object.db_cursor.fetchone()
            if int(test[0]) == 1:
                self._parent.protocol('WM_DELETE_WINDOW', lambda: "do nothing")
                tkMessageBox.showerror("Last test", message = " Sorry but each project needs one unfailed test")
                self._parent.protocol('WM_DELETE_WINDOW', self._parent._gui_quit)
                failed['failed_flag'].set(0)
                break
            if old_flags[i_count]['failed_flag'] != new_flag:
                self.__app._measure_obj.db_object.db_cursor.execute("""UPDATE test SET failed_flag = '%s'
                                                WHERE test_id = '%s'""" % (new_flag,failed['tm_id'].get()))

            i_count +=1


class Mymenu(Tkinter.Menu):
    """class which creates the menu """
    def __init__(self,parent):
        """ constructor to initiliase the menu """
        self.__parent = parent
        Tkinter.Menu.__init__(self,self.__parent)
        self.__create_menu_1()
        self.__create_menu_2()

    def __create_menu_1(self):
        """ create the first menu """
        menu_1 = Tkinter.Menu(self,
            tearoff = '0',
        )
        self.add_cascade(
            columnbreak = '1',
            label = 'Menu',
            menu = menu_1,
        )

        menu_1.add_command(
            label = 'Open db',
            underline = '0',
            command = self.__parent._open_db

        )
        menu_1.add_command(
            label = 'Save db as',
            underline = '0',
            command = self.__parent._save_db

        )

        #self._menu_1.add_command(
        #    label = 'Print',
        #    underline = '0',
        #    command = self.__parent._main_frame._print_it
        #
        #)
        menu_1.add_command(
            label = 'Exit',
            underline = '0',
            command = self.__parent._gui_quit
        )

    def __create_menu_2(self):
        """ create the second menu """
        menu_2 = Tkinter.Menu(self,
            tearoff = '0',
        )
        self.add_cascade(
            label = 'Window',
            menu = menu_2,
        )
        menu_2.add_command(
            label = 'Main',
            underline = '0',
```

```
            command = self.__parent.graph_choice.config
        )
        menu_2.add_command(
            label = 'Configuration',
            underline = '0',
            command = self.__parent.configuration_frame._configuration_frame

        )


class Dialog(Tkinter.Tk):
    """ main Dialog class """

    def __init__(self,tkparent):
        """ constructor for the main Dialog
            initialisation of all main widgets
        """
        Tkinter.Tk.__init__(self)

        #self.sema = threading.BoundedSemaphore(value = 1)
        #self.threads_connected =threads_connected
        self._parent = tkparent
        self._active = "main"

        #self._root = root          #main window

        self._all_measurements = []
        self._button_start = Tkinter.Button(self,
            text = 'Start',
        )
        """ start button to start a connection and get data from servers """

        self._button_stop = Tkinter.Button(self,
            text = 'Stop',
        )
        """ stop the connection and close all open descriptors """

        self.__text_console_output = Tkinter.Text(self,
            height = '0',
            width = '0',
            state = 'disabled'
        )
        """ console output for the gui """



        self.__d_text = {}
        """ text dictionary for the configuration window """

        self.__d_label = {}
        """ label dictionary for the configuration window """

        self.__button_write_config = {}
        """ write config button initialisation """

        self._button_start.configure(
            command = self._button_start_command
        )
        """ configuration of the start button """

        self._button_stop.configure(
            command = self._button_stop_command,
            state = "disabled"
        )
        """ configuration of the stop button """

        self.__text_console_scrollbar = AutoScrollbar(self,)
        """ automatic scrollbar for the text console """
```

```
965             self.__text_console_output.configure(
                    yscrollcommand = self.__text_console_scrollbar.set,

                )
                self.__text_console_scrollbar.configure(
970                 command = self.__text_console_output.yview
                )



975             # Geometry Management
                self.__text_console_scrollbar.grid(
                    in_    = self,
                    column = 2,
                    row    = 1,
980                 columnspan = '1',
                    ipadx = '0',
                    ipady = '0',
                    padx = '0',
                    pady = '0',
985                 rowspan = '2',
                    sticky = 'nsw'
                )

                self._button_start.grid(
990                 in_    = self,
                    column = 0,
                    row    = 1,
                    columnspan = '1',
                    ipadx = '0',
995                 ipady = '0',
                    padx = '0',
                    pady = '0',
                    rowspan = '1',
                    sticky = 'nesw'
1000            )
                self._button_stop.grid(
                    in_    = self,
                    column = 0,
                    row    = 2,
1005                columnspan = '1',
                    ipadx = '0',
                    ipady = '0',
                    padx = '0',
                    pady = '0',
1010                rowspan = '1',
                    sticky = 'nesw',

                )
                self.__text_console_output.grid(
1015                in_    = self,
                    column = 1,
                    row    = 1,
                    columnspan = '1',
                    rowspan = '2',
1020                ipadx = '0',
                    ipady = '0',
                    padx = '0',
                    pady = '0',
                    sticky = 'swen'
1025            )
                self._main_frame = Mainframe(self)
                self._main_frame.grid(
                        in_ = self,
                        column = 0,
1030                    row    = 0,
                        columnspan = '3',
                        ipadx = '0',
                        ipady = '0',
                        padx = '0',
1035                    pady = '0',
```

```
                rowspan = '1',
                sticky = 'nwes'
            )


        self.configuration_frame = Configuration(self)
        self.graph_choice = graph_choice(self,app = tkparent)#._raw_output_frame()
        self.graph_choice.config()
        self._mymenu = Mymenu(self)
        db_name = self._parent._measure_obj.db_object._get_db()
        print db_name
        db_name = db_name.split("/")
        db_name.reverse()
        self.title("SRB Benchmark: "+db_name[0])
        self._dyn = None
        # Resize Behavior
        self.grid_rowconfigure(0, weight = 1, minsize = 40, pad = 0)
        self.grid_rowconfigure(1, weight = 0, minsize = 40, pad = 0)
        self.grid_rowconfigure(2, weight = 0, minsize = 40, pad = 0)
        self.grid_columnconfigure(0, weight = 0, minsize = 40, pad = 0)
        self.grid_columnconfigure(1, weight = 1, minsize = 40, pad = 0)
        self.grid_columnconfigure(2, weight = 0, minsize = 10, pad = 0)
        self.configure(menu = self._mymenu)
        #self.graph_start = threading.Event()

    """ ************** ACTIONS ********************** """

    def _button_start_command(self):
        """ start button command """
        self._button_start.config(state = "disabled")
        self.closed = 0
        ## kick old measurements ##
        self._parent._measure_obj.db_object.get_measurements()
        #self.threads_connected.clear()
        ## start the measurement ##
        i_measurement_return = self._parent._measure_obj.run_measurements()

        ## a server connection failed, stop all other connections ##
        if (i_measurement_return == -1):
            print "no connection reached"

            self._parent._measure_obj.stop_connections()
            self._button_start.config(state = "normal")
        ## all server connected
        else:

            ## set all measurements to zero (for the dynamic table and the graph

            self._all_measurements = []

            graph_supp = myplot.Graph(self._parent._measure_obj.db_object)

            ## start a thread that checks every second if the measurement is still running and stop it if not
            thread.start_new_thread(self._test_stop, ())


            ## set_var controls the test_stop thread. if it is set to one the thread automatically stops
            self._set_var = 0
            # change the state of both buttons


            self._dyn = myplot.dyn_table(self,graph_supp)
            # create a thread for the dynamic graph
            dyn_graph = thread.start_new_thread(self._dyn.config2,())
            self._button_stop.config(state = "normal")



    def _change_text_in_console(self, s_text):
        """ insert function to put text into the console """
```

```
        ## enable the text widget to write
        self.__text_console_output.config(state = "normal")
        ## insert the text at the end
        self.__text_console_output.insert("end", s_text)
        ## disable the console again
        self.__text_console_output.config(state = "disabled")
        ## change the yview to the end
        self.__text_console_output.yview("end")



    def _button_stop_command(self):
        """ stop button command """
        print "pressed stop"
        ## stop the _test_stop thread which looks if the measurement has ended, because the max number of measurement
              is reached
        self._button_stop.config(state = "disabled")


        self._set_var =1

        ## stop the measurement
        self._parent._measure_obj.stop()


        ## restart the main site, to renew the listbox tables
        if self._active == "main":

            self.graph_choice.config()

        ## change the buttons
        self._dyn.stop_table()
        self._button_start.config(state = "normal")

        print "stop button finished"




    def _gui_quit(self):
        """ quit function """
        ## stop the connection to the tables

        if 'disabled' in self._button_start.config('state'):

            self._button_stop_command()

        self._parent._measure_obj.db_object.db_cursor.close()
        self._parent._measure_obj.db_object.db_con.close()
        print "close cursor"

        print "gui_quit just self.quit"
        self.quit()
        ## kill yourself

    def _test_stop(self):
        """ thread to test if the measurement is still running """
        time.sleep(1)
        while 1:
            if self._parent._measure_obj.ganglia_obj._stopevent.isSet():

                if self._set_var == 1:
                    print "thread already stopped"
                    break
                ## press manually the stop button

                self._button_stop_command()
                break
```

```
            time.sleep(1)


    def _open_db(self):
        """ open a different database """
        bad_file = 0
        old_db = self._parent._measure_obj.db_object._get_db()
        while 1:
            self.protocol("WM_DELETE_WINDOW",self._dummy)
            result = tkFileDialog.askopenfilename(filetypes = [("database","*.db")])

            self.protocol('WM_DELETE_WINDOW', self._gui_quit)
            #print result
            if result == "":
                if bad_file == 1:
                    self.protocol("WM_DELETE_WINDOW",self._dummy)
                    tkMessageBox.showerror(title = "Error", message = "take the old database")
                    self.protocol('WM_DELETE_WINDOW', self._gui_quit)
                    self._parent._measure_obj.db_object.set_new_db(old_db)
                    result = old_db
                break
            if self._parent._measure_obj.db_object.set_new_db(result) == 0:
                self.graph_choice.config()
                break
            else:
                self.protocol("WM_DELETE_WINDOW",self._dummy)
                tkMessageBox.showerror(title = "Error", message = "Bad database file")

                self.protocol('WM_DELETE_WINDOW', self._gui_quit)

                bad_file = 1
        result = result.split("/")
        result.reverse()
        self.title("SRB Benchmark: "+result[0])



    def _save_db(self):
        """ function to save the actual database in another file """
        db = self._parent._measure_obj.db_object._get_db()
        db_name = db.split("/")
        db_name.reverse()
        self.protocol("WM_DELETE_WINDOW",self._dummy)
        result = tkFileDialog.asksaveasfilename(filetypes = [("database","*.db")], title = "Save "+db_name[0]+" AS")

        self.protocol('WM_DELETE_WINDOW', self._gui_quit)

        if result != "":
            shutil.copyfile(db,result)
            if self._parent._measure_obj.db_object.set_new_db(result) == -1:
                self.protocol("WM_DELETE_WINDOW",self._dummy)
                tkMessageBox.showerror(title = "Error", message = "Bad database file")
                self.protocol('WM_DELETE_WINDOW', self._gui_quit)

                bad_file = 1
                self._parent._measure_obj.db_object.set_new_db(db)
            else:
                result = result.split("/")
                result.reverse()
                self.title("SRB Benchmark: "+result[0])

    def _dummy(self,event = None):
        """ dummy function, to do nothing """
        return 'break'

    """ ********** CONFIGURATION FRAME ********** """
class Configuration:
    """ class for the configuration frame, which sets and writes a new config file """
    def __init__(self,parent):
        self._parent = parent
        self._main_frame = parent._main_frame
```

```
        self._measure_obj = parent._parent._measure_obj
        self.__d_text = {}
        self.__d_label = {}

    def _configuration_frame(self, server_number = None):
        """ set the frame for the configuration vision """

        self._active = "config"

        if server_number != None:
            self.server_number = int(server_number)
        self._main_frame.config()

        l_config_keys = []
        l_config_keys[1:] = self._measure_obj.get_config().keys()
        l_config_keys.sort()

        counter = 1
        d_config = self._measure_obj.get_config()

        if server_number == None:
            self.server_number = int(d_config['server.quantity'])
        for s_key in l_config_keys:
            if s_key == "server.quantity":
                row_number = 0
            else:
                row_number = counter
            if s_key.startswith("server.") == False:
                self.__d_label[s_key] = Tkinter.Label(self._main_frame._frame, text = s_key)
                self.__d_label[s_key].grid(
                column = 1,
                row    = row_number,
                columnspan = '1',
                ipadx = '0',
                ipady = '0',
                padx = '0',
                pady = '0',
                rowspan = '1',
                sticky = 'nwes'
                )
            else:
                counter -= 1

            if s_key == "server.quantity":
                self.__d_label[s_key] = Tkinter.Label(self._main_frame._frame, text = s_key)
                self.__d_label[s_key].grid(

                column = 1,
                row    = row_number,
                columnspan = '1',
                ipadx = '0',
                ipady = '0',
                padx = '0',
                pady = '0',
                rowspan = '1',
                sticky = 'nwes'
                )
                self.__quantity_listbox = Tkinter.Listbox(self._main_frame._frame, height = 2, width = 10, bg = "white
                    ")
                self.__quantity_listbox.grid(
                column = 2,
                row    = row_number,
                columnspan = '1',
                ipadx = '0',
                ipady = '0',
                padx = '0',
                pady = '0',
                rowspan = '1',
                sticky = 'nwes'
                )
                for server in range(1, 10):
```

```
                    self.__quantity_listbox.insert("end", server)

1320            select_scrollbar = Tkinter.Scrollbar(self._main_frame._frame)
                self.__quantity_listbox.configure(
                    yscrollcommand = select_scrollbar.set,


1325                )
                self.__quantity_listbox.bind("<Button-1>", self._parent._dummy)
                self.__quantity_listbox.bind("<Double-Button-1>", self._cur_selec)

                select_scrollbar.configure(
1330               command = self.__quantity_listbox.yview,
                    )

                select_scrollbar.grid(
                    in_     = self._main_frame._frame,
1335                column  = 3,
                    row     = row_number,
                    columnspan = '1',
                    ipadx = '0',
                    ipady = '0',
1340                padx = '0',
                    pady = '0',
                    rowspan = '1',
                    sticky = 'nws',

1345            )
            elif s_key.startswith("server."):
                """do nothing """
            elif s_key == "measurement.srb":
                self.__d_text[s_key] = Tkinter.StringVar()
1350            self.__d_text[s_key].set(d_config[s_key])
                srb_toggle= Tkinter.Checkbutton(self._main_frame._frame, variable = self.__d_text[s_key], onvalue = "1
                    ", offvalue = "0")
                srb_toggle.grid(
                    column = 2,
                    row    = row_number,
1355                columnspan = '1',
                    ipadx = '0',
                    ipady = '0',
                    padx = '0',
                    pady = '0',
1360                rowspan = '1',
                    sticky = 'nwes'
                )
            else:
                self.__d_text[s_key] = Tkinter.Entry(self._main_frame._frame, width = 50, bg = "white")
1365            self.__d_text[s_key].insert("insert", d_config[s_key])
                self.__d_text[s_key].grid(
                    column = 2,
                    row    = row_number,
                    columnspan = '1',
1370                ipadx = '0',
                    ipady = '0',
                    padx = '0',
                    pady = '0',
                    rowspan = '1',
1375                sticky = 'nwes'
                )

            self._main_frame._frame.grid_rowconfigure(counter, weight = 0, minsize = 10, pad = 0)
            counter = counter +1
1380
        for server in range(1, self.server_number+1):

            for component in range(1, 3):
                if component == 1:
1385                server_key = "server.host_"+str(server)
                elif component == 2:
                    server_key = "server.port_"+str(server)
```

```
            # else:
             #     server_key = "server.poll_time_"+str(server)

             self.__d_label[server_key] = Tkinter.Label(self._main_frame._frame, text = server_key)
             self.__d_label[server_key].grid(
                 column = 1,
                 row    = counter,
                 columnspan = '1',
                 ipadx = '0',
                 ipady = '0',
                 padx = '0',
                 pady = '0',
                 rowspan = '1',
                 sticky = 'nwes'
                 )
             self.__d_text[server_key] = Tkinter.Entry(self._main_frame._frame, width = 50, bg ="white")
             self.__d_text[server_key].grid(
                 column = 2,
                 row    = counter,
                 columnspan = '1',
                 ipadx = '0',
                 ipady = '0',
                 padx = '0',
                 pady = '0',
                 rowspan = '1',
                 sticky = 'nwes'
             )
             if d_config.has_key(server_key):
                 self.__d_text[server_key].insert("insert", d_config[server_key])
             self._main_frame._frame.grid_rowconfigure(counter, weight = 0, minsize = 10, pad = 0)
             counter += 1

        self.__button_set_config = Tkinter.Button(self._main_frame._frame, text = 'Set', command = self.
            __button_set_command)
        self.__button_set_config.grid(
            column = 1,
            row    = counter,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nws'
        )
        self.__button_write_config = Tkinter.Button(
            self._main_frame._frame,
            text = 'Write Config',
            command = self.__button_write_config_command
        )

        self.__button_write_config.grid(
            column = 2,
            row    = counter,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            pady = '0',
            rowspan = '1',
            sticky = 'nws'
        )
        self._main_frame._frame.grid_columnconfigure(2, weight = 1, minsize = 10, pad = 0)

        self._main_frame._canvas.create_window(0, 0, anchor = "nw", window = self._main_frame._frame)
        self._main_frame._frame.update_idletasks()

        self._main_frame._canvas.config(scrollregion = self._main_frame._canvas.bbox("all"))
```

```
    def __button_set_command(self):
        """ set command for the configuration set button """

        test = self._parent._button_start.config('state')
        if 'disabled' in test:
            self._parent.protocol("WM_DELETE_WINDOW",self._parent._dummy)
            tkMessageBox.showerror(title = "Error", message = "Cannot set the config during measurement")
            self._parent.protocol('WM_DELETE_WINDOW', self._parent._gui_quit)
        else:
            d_config = {}
            for content in self.__d_text:

                try:
                    d_config[content] = self.__d_text[content].get()
                except Tkinter.TclError:
                    print Exception
                    d_config[content] = "None"

                d_config[content] = d_config[content].replace("""'""","""  """)
                d_config[content] = d_config[content].replace('"',' ')
                d_config[content] = d_config[content].strip()

            for x in range(0,self.server_number):
                if d_config['server.host_'+str(x+1)].find("localhost") != -1 or d_config['server.host_'+str(x+1)].find
                    ("127.0.0.1") != -1:
                    self._parent.protocol("WM_DELETE_WINDOW",self._parent._dummy)
                    tkMessageBox.showerror(title = "Error", message = "bad server")
                    self._parent.protocol('WM_DELETE_WINDOW', self._parent._gui_quit)
                    return 0
            # print d_config['test.name']
            if d_config['test.name'] == "" or d_config['project.name'] == "" or d_config['tester.forename'] == "" \
            or d_config['tester.surname'] == "" or d_config['measurement.quantity'] == "" or \
            d_config['measurement.poll_time'] == "" or d_config['measurement.application'] == "":
                self._parent.protocol("WM_DELETE_WINDOW",self._parent._dummy)
                tkMessageBox.showerror(title = "Error", message = "necessary configuration entries are missing")
                self._parent.protocol('WM_DELETE_WINDOW', self._parent._gui_quit)
            else:
                d_config['server.quantity'] = self.server_number
                self._measure_obj.set_config(d_config)
                self._configuration_frame(self.server_number)

    def __button_write_config_command(self):
        """ command for the write button on the configuration frame"""
        self._parent.protocol("WM_DELETE_WINDOW",self._parent._dummy)
        write_to = tkSimpleDialog.askstring("Write config to","test:",  initialvalue = self._parent._parent._cparser.
            get_config_file())
        self._parent.protocol('WM_DELETE_WINDOW', self._parent._gui_quit)

        if write_to != None:

            self.__button_set_command()

            test = self._parent._button_start.config('state')

            if self._parent._parent._cparser.write(write_to, self._measure_obj.get_config()) == -1:
                tkMessageBox.showerror(title = "Write Error", message = "Cannot write the config: File Error?")



    def _cur_selec(self, event = None):
        """ get the current selection of the listbox """
        i = self.__quantity_get_index(event)
        if i > -1:
            self.__quantity_listbox.select_clear(0, 'end')
            self._configuration_frame(i+1)
            self.__quantity_listbox.see(i)
            self.__quantity_listbox.selection_set(i)


    def __quantity_get_index(self, event):
```

```
        """ help the _cur_selec function to get the actual position in the listbox """
        x_org = self.__quantity_listbox.winfo_rootx()
        y_org = self.__quantity_listbox.winfo_rooty()
        x_new = event.x_root - x_org
        y_new = event.y_root - y_org
        index = self.__quantity_listbox.index('@' + str(x_new) + ',' + str(y_new))
        box = self.__quantity_listbox.bbox(index)
        if not box or y_new > box[1] + box[3]:
            return -1
        else:
            return index
```

B.8 myplot.py

```
#!/usr/local/bin/python

""" MYPLOT creates the Graph-window and creates the graphs """

__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "15.12.2005"
__version__ = "0.1"
__revision__ = "1.0"


from Graphs import *

import time
import tkFileDialog
import random
import tooltip
import re

class Graph(Tk):
    """ class which is able to plot the measurement graphs, all data will be set up for the Graphs.py """

    def __init__(self,db_object):
        """ constructor to initialize the Dialog with its Slider and Buttons """

        Tk.__init__(self)
        # Don't allow the window to close before everything is initiliazed, (if not could it could cause thread
            problems)
        self.protocol('WM_DELETE_WINDOW', self.__dummy)
        #database object
        self.__db_object = db_object
        #create a statusbar object
        self.__statusbar = StatusBar(self)
        #set the title of the window
        self.title('simple bar graph')
        # create the frame for the graph
        self.__f1 = Frame(self)

        #dummy settings
        self.__parent = None
        self.__balloonhelp = None
        self.__showed = None
        self.__begin = None
        self.__end = None

        # create the slider for centering
        self.__scale_begin = Scale(self, orient=HORIZONTAL, state = 'disabled')
        # create the slider for zooming
        self.__scale_zoom = Scale(self, orient=HORIZONTAL, state = 'disabled')
        #create the labels for the slider
        zoom_label = Label(self, text = "Zoom in/out:")
        value_label = Label(self, text = "Center on:")
```

```
        # create the button which toggles the legend on and off
        self.__b_legend = Button(self, text = 'Legend')

        # create the button which allows the storing of the graph as a postscript
        self.__b_print = Button(self, text = 'Save graph')
        # bind the button to the tooltip function
        self.__b_print.bind("<Enter>",lambda ev,self = self,\
                            ht = """ Save the graph in a Postscript File """,
                              tt = """ Save graph as ps """:\
                              self.on_enter(ev,ht,tt))
        # set the leave function to the print button, which destroys the tooltip
        self.__b_print.bind("<Leave>",self.on_leave)

        #create the Listbox with the possible graphs
        self.__listb = Listbox(self, height = 5, bg = "white")          # the listbox with the keys to choose
        self.__listb.bind("<Enter>",lambda ev,self = self,\
                            ht = """ Double - Click the Value you want to see in the graph """,
                              tt = """ Data list """:\
                              self.on_enter(ev,ht,tt))
        self.__listb.bind("<Leave>",self.on_leave)

        # setup the columns between the widgets, so they can become bigger
        self.grid_columnconfigure(3, weight = 1)
        self.grid_columnconfigure(6, weight = 1)
        self.grid_columnconfigure(8, weight = 1)

        # dummy settings
        self.__graph = None
        self.__key_list = []
        #self.__full_list = {}

        # scrollbar for the listbox
        select_scrollbar = Scrollbar(self)
        self.__listb.configure(
            yscrollcommand = select_scrollbar.set
        )
        # stop the normal click on the Listbox
        self.__listb.bind("<Button-1>", self.__dummy)
        # double click and a new graph will be seen

        # bing the scollbar to the listbox
        select_scrollbar.configure(
            command = self.__listb.yview,
        )

        #self.table_length = 20
        #self.longest_table = None

        # start settings:

        # sets the active graph
        self.__active = None

        self.__length = 0
        self.__color_values = {}
        self.__used_color = 0

##### SET THE GRID LOCATIONS OF THE WINDOW FOR EVERY WIDGET #############
        self.__b_print.grid(
            column  = 7,
            row     = 1,
            sticky = "swe"
        )

        self.__listb.grid(
            column  = 1,
            row     = 1,
            columnspan = 1,
            sticky  = "we",
            rowspan = '2',
```

```
        )
        zoom_label.grid(
            in_     = self,
            column  = 4,
            row     = 2,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            rowspan = '1',
            sticky = 'ws',
        )
        value_label.grid(
            in_     = self,
            column  = 4,
            row     = 1,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            rowspan = '1',
            sticky = 'ws',
        )
        select_scrollbar.grid(
            in_     = self,
            column  = 2,
            row     = 1,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            rowspan = '2',
            sticky = 'nws',
        )

        self.__b_legend.grid(
            column  = 7,
            row     = 2,
            sticky = "swe"
        )
        self.__scale_begin.grid(
            in_     = self,
            column  = 5,
            row     = 1,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            rowspan = '1',
            sticky = 'nws',
        )

        self.__scale_zoom.grid(
            in_     = self,
            column  = 5,
            row     = 2,
            columnspan = '1',
            ipadx = '0',
            ipady = '0',
            padx = '0',
            rowspan = '1',
            sticky = 'nws',
            )

        self.__statusbar.grid(
            column = 0,
            row     = 3,
            columnspan = 8,
            sticky = "news"
        )
```

```
######### END OF GRID LOCATION SETTING ######################

        # set the "Ready" text to the statusbar
        self.__statusbar.set()

        ## set the tool tips for the slider
        self.__scale_begin.bind("<Enter>",lambda ev,self = self,\
                                ht = """ This slider sets the time value in which the zoom slider will zoom""",
                                  tt = """ Value to zoom """:\
                                  self.on_enter(ev,ht,tt))
        self.__scale_begin.bind("<Leave>",self.on_leave)
        self.__scale_zoom.bind("<Enter>",lambda ev,self = self,\
                                ht = """ This slider sets number of values, which shall be shown in the graph""",
                                tt = """ Zoom slider""":\
                                  self.on_enter(ev,ht,tt))
        self.__scale_zoom.bind("<Leave>",self.on_leave)
        # start value for the legend (no legend at the beginning)
        self.__legend_changer = 0
        self.__itersrbserv_comb = {}
        self.__view = "single"

    def on_enter(self,event,helptext,tooltext):
        """ starts the tooltip and sets the statusbar with the current widget information """

        if helptext:
            self.__statusbar.set(text = helptext)
        if self.__balloonhelp != None and self.__showed != None:
            self.__balloonhelp.destroy()
            self.__showed = None

        if tooltext:
            y = event.widget.winfo_rooty() -25
            self.__balloonhelp = tooltip.ToolTip(tooltext,event.widget.winfo_rootx(),y)
            self.after_id = self.after(300,self.on_after)


    def on_after(self):
        """ shows the tooltip after a particular time"""

        if self.__balloonhelp != None:
            self.__showed = 1
            self.__balloonhelp.show()


    def on_leave(self,event=None):
        """ stops the tooltip and resets the statusbar """

        if self.__showed != None and self.__balloonhelp != None:
            self.__balloonhelp.destroy()
            self.__showed = None
        else:
            self.__balloonhelp = None
        self.__statusbar.set()


    def callback(self,event):
        """ algorithm which calculates the zooming of the graph and sets the slider """

        if self.__itersrbserv_comb != {} and (not 'disabled' in self.__scale_begin.config('state')):

            #get the values from the slider
            middle_value = self.__scale_begin.get()
            number_of_values = self.__scale_zoom.get()

            # maximum zoom out value
            max_num_of_values = int(float(self.__scale_zoom['from']))

            # max value of the time_scale
            max_time = self.__scale_begin['to']

            ## calculate the beginning and the end of the zooming
```

```
            percent = float(middle_value)/float(max_time)
            starting_value = int ( percent * float(max_num_of_values))

            side_values_left=side_values_right = number_of_values/2
            if number_of_values%2 == 1:
                side_values_right +=1

            if int(starting_value + side_values_right) > max_num_of_values:

                end = max_num_of_values
                side_values_left += starting_value + side_values_right - max_num_of_values
                begin = starting_value - side_values_left
            elif (starting_value - side_values_left) < 0:
                begin = 0
                end = starting_value + side_values_right
                end += abs(starting_value - side_values_left)
            else:
                begin   = starting_value - side_values_left
                end     = starting_value + side_values_right
            try:
                self.config(active = self.__active, begin = begin, end =end)
            except:
                print "refresh of window and doubleclick on window crushed"


    def _cur_selec(self, event = None):
        """ returns the current value of table choice,
        is used to create a new graph"""

        i = self.__quantity_get_index(event,self.__listb)
        if i > -1:
            self.__listb.select_clear(0, 'end')

            self.__listb.see(i)
            self.__listb.selection_set(i)

            #if self.__view == "single":
            self.config(active = self.__listb.get(i))
            #else:
             #    self.config(active = 'M.'+self.__listb.get(i))


    def __dummy(self,event = None):
        """ just to do nothing """

        return 'break'


    def __quantity_get_index(self, event,listb):
        """ calculates the current position in the table listbox and returns it to _cur_selec """

        x_org = listb.winfo_rootx()
        y_org = listb.winfo_rooty()
        x_new = event.x_root - x_org
        y_new = event.y_root - y_org
        index = listb.index('@' + str(x_new) + ',' + str(y_new))
        box = listb.bbox(index)

        if not box or y_new > box[1] + box[3]:
            return -1
        else:
            return index


    def close_me(self):
        """ close the graph class and destroy it """

        if self.__balloonhelp != None and self.__showed !=None:
            self.__balloonhelp.destroy()
        if self.__parent != None:
            print "I have a parent"
```

```
            self.__parent._dead_graph = 0
            if 'disabled' in self.__parent._parent._button_start.config('state'):
                self.__parent._parent._button_stop_command()
        else:
            print "stop the graph myself"
            self.destroy()


    #def set_list(self,list  = []):

     #   self.list = list

    def print_it(self):
        """ function to print the graph in a postscript file """

        self.protocol('WM_DELETE_WINDOW', self.__dummy)
        result = tkFileDialog.asksaveasfilename(filetypes = [("postscript","*.ps")], title = "Save graph as", parent =
            self)
        self.protocol('WM_DELETE_WINDOW', self.close_me)
        if result != "":
            self.__graph.canvas.postscript(file=result, colormode = "color", rotate = "true")
            print " print it to %s" % result


    def legend_toggle(self):
        """ toggle the legend on or off """

        if self.__legend_changer == 0:
            self.__legend_changer = 1
        else:
            self.__legend_changer = 0
        if self.__zoom_start != None:
            self.config(active = self.__active,begin = self.__zoom_start, end = self.__zoom_end)
        else:
            self.config(active = self.__active)


    def colors(self):
        """ sets a particular color to a graph (6 standard colors than random colors) """

        color_values = ["black", "red", "green", "blue", "cyan", "yellow", "magenta"]
        color = ""
        if self.__used_color+1 > len(color_values):
            while len(color) < 6:
                color_part = hex(random.randint(0,255))
                if len(color_part)<4:
                    color_part='0'+color_part[2:]
                color += color_part[2:]
        else:
            self.__used_color +=1
            return color_values[self.__used_color-1]

        self.__used_color +=1
        return '#'+color


    def mousemov(self,event):
        """ function for a left double click in the graph ( automatically zooming 2x on this x-position)"""

        x_value = self.__graph.canvas.canvasx(event.x)
        width = float(self.__graph.canvas.cget('width'))
        x_value = x_value -0.1*width
        width = width - 0.16*width

        if (width > x_value) and (x_value >= 0.0):
            if self.__end ==None:
                self.__scale_begin.set(int((x_value/width)*float(self.__scale_begin.cget('to'))))
            else:
                self.__scale_begin.set(int((x_value/width)*self.__end)+int((1.0-(x_value/width))*self.__begin))
            self.__scale_zoom.set(int((0.5)*float(self.__scale_zoom.get())))
            self.callback(event)
```

```
405
    def mousemov_back(self,event):
        """ function for a right double click in the graph ( automatically zooming 1/2x on this x-position)"""

410     self.__scale_zoom.set(int((2.0)*float(self.__scale_zoom.get())))
        self.callback(event)


    def __create_view_multiple(self,list):
415     """ Create the view for more than one test (reused test)"""

        # set the view to multiple view
        self.__view = "multiple"
        self.__itersrbserv_comb = {}
420     count = 0
        self.start_time = {}
        self.multilegend = {}

        # get some information for the legend, so the graphs can be relocated
425     # furthermore store the data in a dictionary for the graphs
        for list_elem in list:

            self.__db_object.lock_it()
            self.__db_object.db_cursor.execute("""SELECT I.time, H.hostname, S.srb_port,I_S.is_id FROM
430                                             iteration_srbserver AS I_S, srbserver AS S, iteration AS I, host
                                                    AS H, test AS T
                                             where I_S.is_id = %s AND I_S.srbserver_id = S.srbserver_id
                                             AND S.host_id = H.host_id AND I.iteration_id = I_S.iteration_id
                                             AND I.test_id = T.test_id  """ % list_elem["M.is_id"])#, T.name,

435         test_time = self.__db_object.db_cursor.fetchone()
            self.__db_object.rlock()
            is_id = str(test_time[3])
            if not self.start_time.has_key(is_id):
                self.start_time[is_id] = list_elem["M.time"]
440             self.multilegend[is_id] = (test_time[0],test_time[1],test_time[2])
            if count == 0:
                count = 1
                keys = list_elem.keys()
                s = re.compile("(.*)(time|_id$)")
445             for x in keys:
                    if not s.match(x):
                        self.__itersrbserv_comb[x[2:]] = {}
            for x in keys:
                if not s.match(x):
450                 if not self.__itersrbserv_comb[x[2:]].has_key(is_id):
                        self.__itersrbserv_comb[x[2:]][is_id] = []
                    self.__itersrbserv_comb[x[2:]][is_id].append((time.mktime(time.strptime(list_elem['M.time'],'%Y-%m
                        -%d %H:%M:%S'))\
                                                    -time.mktime(time.strptime(self.start_time[is_id],'%Y-%m-%d %H:%M
                                                        :%S'))\
                                                    ,float(list_elem[x])))
455
            self.__db_object.lock_it()
            self.__db_object.db_cursor.execute("SELECT A.* FROM application AS A, measurement AS M WHERE A.
                measurement_id = %s \
                                        and M.measurement_id = A.measurement_id" %list_elem["M.measurement_id"])
            applications = self.__db_object.db_cursor.fetchall()
460         self.__db_object.rlock()

            # store the values for the particular time in is_id
            for x in applications:
                for key, item in x.items():
465                 if key != "A.name" and key != "A.command" and key != "A.application_id" and key != "A.
                        measurement_id":
                        if not self.__itersrbserv_comb.has_key(x['A.name']+"_"+key[2:]):
                            self.__itersrbserv_comb[x['A.name']+"_"+key[2:]] = {}
                        if not self.__itersrbserv_comb[x['A.name']+"_"+key[2:]].has_key(is_id):
                            self.__itersrbserv_comb[x['A.name']+"_"+key[2:]][is_id] = []
```

```
                    self.__itersrbserv_comb[x['A.name']+"_"+key[2:]][is_id].append((time.mktime(time.strptime(
                        list_elem['M.time'],'%Y-%m-%d %H:%M:%S'))\
                                    -time.mktime(time.strptime(self.start_time[is_id],'%Y-%m-%d %H:%M
                                        :%S'))\
                                    ,float(item)))

        count = 0

        # setup the listbox with possible graphs
        if self.__listb.size() == 0:
            for x in self.__itersrbserv_comb.keys():
                self.__key_list.append(x)
            self.__key_list.sort()
            for x in self.__key_list:
                self.__listb.insert("end",x)
            active = x

        # store the current graph in self.__active
        if self.__active == None:
           # self.lists = self.__itersrbserv_comb[active]
            self.__active = active
            #self.title(active)
       #else:
            #self.lists = self.__itersrbserv_comb[self.__active]
            #self.title(self.__active)



    def __create_view_single(self,list):
        """create the view for only one test"""

        self.__view = "single"
        self.__db_object.lock_it()
        self.__db_object.db_cursor.execute("SELECT A.*,M.time FROM application AS A, measurement AS M WHERE A.
            measurement_id = %s and \
                                    M.measurement_id = A.measurement_id" %list[0]["M.measurement_id"])
        applications = self.__db_object.db_cursor.fetchall()
        self.__db_object.rlock()
        self.__itersrbserv_comb = {}
        # create the dictionary with possible graphs
        for x in applications[0].keys():
            if not re.match("(.*)(command|name|time|id)$",x):
                self.__itersrbserv_comb[x[2:]] = {}

        for x in applications:
            self.__db_object.lock_it()
            self.__db_object.db_cursor.execute("SELECT A.*,M.time FROM application AS A, measurement AS M WHERE M.
                is_id = %s and \
                                    M.measurement_id = A.measurement_id and A.name = '%s' " % (list[0]["M.
                                        is_id"],x["A.name"]))
            apps = self.__db_object.db_cursor.fetchall()
            self.__db_object.rlock()
            # create the lists for the applications
            for y in self.__itersrbserv_comb.keys():
                self.__itersrbserv_comb[y][x["A.name"]] = []
            # store the beginning of the measurement in start_time
            self.start_time = apps[0]["M.time"]
            for app_item in apps:
                for y in self.__itersrbserv_comb.keys():
                    #store the value for the particular time in the tm_id dictionary
                    self.__itersrbserv_comb[y][app_item["A.name"]].append((time.mktime(time.strptime(app_item["M.time"
                        ],'%Y-%m-%d %H:%M:%S'))- \
                                                time.mktime(time.strptime(self.start_time,'%Y-%m-%d %H:%
                                                    M:%S')), \
                    float( app_item["A."+y])))

        count = 0
        meas_keys = []

        # the lists for the graphs will be generated and stored in the dictionary tm_id
        for list_elem in list:
```

```
            if count == 0:
                count = 1
                for key,value in list_elem.items():

                    if key.find("server") != -1:
                        client_key = re.sub("server","client",key)
                        if list_elem.has_key(client_key):
                            name = re.sub("M.server_","",key)
                            meas_keys.append(name)
                            self.__itersrbserv_comb[name] = {}
                            self.__itersrbserv_comb[name]["server"] =[]
                            self.__itersrbserv_comb[name]["client"] =[]

            # append the values for server cpu load and client cpu load
            for measkey in meas_keys:
                self.__itersrbserv_comb[measkey]["client"].append(( time.mktime(time.strptime(list_elem["M.time"],'%Y
                    -%m-%d %H:%M:%S'))- \
                                                                    time.mktime(time.strptime(self.start_time,'%Y-%m-%d %H:%
                                                                        M:%S')), \
                    float(list_elem['M.client_'+measkey])))
                self.__itersrbserv_comb[measkey]["server"].append((time.mktime(time.strptime(list_elem["M.time"],'%Y-%
                    m-%d %H:%M:%S'))- \
                                                                    time.mktime(time.strptime(self.start_time,'%Y-%m-%d %H:%
                                                                        M:%S')), \
                    float(list_elem['M.server_'+measkey])))

        active = measkey
        if self.__listb.size() == 0:
            for x in self.__itersrbserv_comb.keys():
                self.__key_list.append(x)
                self.__key_list.sort()
            for x in self.__key_list:
                self.__listb.insert("end",x)
        if self.__active == None:
          #  self.lists = self.__itersrbserv_comb[active]
            self.__active = active
            #self.title(active)
        #else:
         #   self.lists = self.__itersrbserv_comb[self.__active]
            #self.title(self.__active)


    def __show_graphs(self,begin = None, end = None):
        """ show the graph for a single test or multiple tests """

        if self.__graph != None:
            self.__graph.destroy()


        # create the legend
        legend = {}
        time_val = []
        if self.__view == "multiple":
            print "multilegend:",self.multilegend
            legend['machine'] = []
            legend['port'] = []
            legend['time'] = []
        else:
            legend[self.__active] = []
        liste = []
        self.__color_values = {}
        self.__used_color = 0


        for x in self.__itersrbserv_comb[self.__active].keys():
            if not self.__color_values.has_key(x):
                self.__color_values[x] = self.colors()
                if self.__view == "multiple":
                    legend['time'].append((self.multilegend[x][0],self.__color_values[x]))
                    legend['machine'].append((self.multilegend[x][1],self.__color_values[x]))
                    legend['port'].append((self.multilegend[x][2],self.__color_values[x]))
```

```
                else:
                    legend[self.__active].append((x,self.__color_values[x]))
            # legend finished

            # create the list for the x-axis
            self.__id_list = []

            #any_key = self.__itersrbserv_comb[self.__active].keys()
            self.__length = 0

            for x in self.__itersrbserv_comb[self.__active].keys():
                if len(self.__itersrbserv_comb[self.__active][x]) > self.__length:
                    self.__length = len(self.__itersrbserv_comb[self.__active][x])
                    id = x
            for x in self.__itersrbserv_comb[self.__active][id]:
                self.__id_list.append((x[0],str(x[0])))

            if self.__legend_changer == 0:
                legend = None

            #print legend

            #create the graph_objects
            if begin != None:

                self.__graph = GraphBase(self.__f1, 500, 500, liste = (self.__id_list[begin:end]),\
                                         relief=SUNKEN, border=2, y_label = self.__active,\
                                         legend = legend)
                self.__end = self.__id_list[end-1][0]
                self.__begin = self.__id_list[begin][0]
            else:
                self.__graph = GraphBase(self.__f1, 500, 500, liste = self.__id_list,\
                                         relief=SUNKEN, border=2, y_label = self.__active,\
                                         legend = legend)


            #set the slider to the possible values
            if len(self.__itersrbserv_comb[self.__active][id]) > 2:
                self.__scale_zoom.configure(
                    from_ = len(self.__itersrbserv_comb[self.__active][id]),
                    to = 2,
                    state = 'active'
                )
                self.__scale_begin.configure(
                    to = self.__itersrbserv_comb[self.__active][id][-1][0],
                    state = 'active'
                )
            if begin == None:
                self.__scale_zoom.set(len(self.__itersrbserv_comb[self.__active][id]))

            # draw the lines
            for x in self.__itersrbserv_comb[self.__active].keys():
                if begin == None:

                    if len(self.__itersrbserv_comb[self.__active][x]) == 1:
                        short_line = [(-0.1,-1.0)]
                        short_line.append(self.__itersrbserv_comb[self.__active][x][0])
                        lines = GraphLine(short_line, color = self.__color_values[x],\
                                          width = 2, smooth = 0)
                    else:
                        lines = GraphLine(self.__itersrbserv_comb[self.__active][x],\
                                          color = self.__color_values[x], width = 2,smooth = 0)
                else:

                    if begin+2 > len(self.__itersrbserv_comb[self.__active][x]):
                        if begin == 0:
                            if len(self.__itersrbserv_comb[self.__active][x]) == 1:
                                short_line = [(-0.1,-1.0)]
                                short_line.append(self.__itersrbserv_comb[self.__active][x][0])
                                lines = GraphLine(short_line, color = self.__color_values[x],\
                                          width = 2, smooth = 0)
```

```
                        else:
                            lines = GraphLine(self.__itersrbserv_comb[self.__active][x],\
                                    color = self.__color_values[x], width = 2,smooth = 0)
                    else:
                        continue

                else:
                    lines = GraphLine(self.__itersrbserv_comb[self.__active][x][begin:end],\
                                    color = self.__color_values[x], width = 2, smooth = 0)



            liste.append(lines)
        return liste


    def config(self,list = [],parent = None, active = None, table_num = None, begin = None, end = None):
        """ configuration of the graph --> creates if necessary and shows the graph """

        # test if it is a dynamic graph or a static graph
        if parent != None:
            self.__parent = parent

        # set the close_me function to the exit (x)- Button
        self.protocol("WM_DELETE_WINDOW", self.close_me)
        self.__scale_begin.bind("<ButtonRelease-1>",self.__dummy())
        self.__scale_zoom.bind("<ButtonRelease-1>",self.__dummy())
        self.__listb.bind("<Double-Button-1>", self.__dummy())

        # change the table for the graph if someone clicked on the listbox
        if active != None:
            self.__active = active
            #self.title(self.__active)
        # destroy the frame with the graph
        self.__fl.destroy()
        #create a new frame
        self.__fl = Frame(self)
        self.__fl.grid(
            column  = 0,
            row     = 0,
            columnspan = 9,
            sticky  = "news",
                 )
        self.grid_rowconfigure(0, weight = 1, pad = 0)
        self.grid_columnconfigure(0, weight = 1, pad = 0)

        # safe the zoom begin and end
        self.__zoom_start = begin
        self.__zoom_end = end

        # create a new measurement graph
        if active == None:
            self.__key_list = []

            self.__id_list = {}
            self.__itersrbserv_comb = {}

            ##sort the measurements after time_id
            for list_elem in list:
                if self.__itersrbserv_comb.has_key(list_elem['M.is_id']) == False:
                    print list_elem['M.is_id']
                    self.__itersrbserv_comb[list_elem['M.is_id']] = []


            # if there is more than one test create a view for multiple tests
            if len(self.__itersrbserv_comb) > 1:
                active = self.__create_view_multiple(list)

            else:
                # else create a view for a single test
```

```
                self.__create_view_single(list)
            liste = self.__show_graphs()

        else:
            #show an already stored graph
            liste = self.__show_graphs(begin,end)

        graphObject = GraphObjects(liste)
        self.title(self.__active)
        self.__graph.draw(graphObject, 'automatic', 'automatic')
        self.__graph.canvas.bind("<Double-Button-1>",self.mousemov)
        self.__graph.canvas.bind("<Double-Button-3>",self.mousemov_back)
        self.__graph.canvas.bind("<Enter>",lambda ev,self = self,\
                                 ht = """DoubleClick Left Mouse Button to zoom in and Right Button to zoom out""",
                                   tt = None:\
                                   self.on_enter(ev,ht,tt))
        self.__graph.canvas.bind("<Leave>",self.on_leave)
        self.__graph.pack(fill=BOTH,expand=YES)
        self.__scale_begin.bind("<ButtonRelease-1>",self.callback)
        self.__scale_zoom.bind("<ButtonRelease-1>",self.callback)
        self.__listb.bind("<Double-Button-1>", self._cur_selec)


        self.__b_legend.configure(command = self.legend_toggle)
        self.__b_print.configure(command = self.print_it)


class StatusBar(Frame):
    """ class which creates a statusbar with one label field """

    def __init__(self,parent):
        """ initialize the graph """

        Frame.__init__(self,parent)
        self.label = Label(self, bd = 1, relief = "sunken", anchor = W)
        self.label.pack(side = "left", fill = "x", expand = "yes", padx = 2, pady = 1)

    def set(self, text = "Ready"):
        """ change the text of the label in the statusbar """

        self.label['text'] = text
        self.label.update_idletasks()


class dyn_table:
    """ class which collects the measurements and draws a dynamic graph """
    def __init__(self,parent,plot_object):
        """ constructor to setup a new graph """
        self.__plot_object = plot_object
        self._parent = parent

        #self.list = []
        if self.__plot_object == None:
            self._dead_graph = 1
        else:
            self._dead_graph = 0


    def config2(self):
        self._show_table()

    def stop_table(self):
        if self._dead_graph == 0:
            self._dead_graph = 1
        #if self.plot_object != None:
        try:
            self.__plot_object.destroy()
        except:
            "Graph already dead"
            #self.plot_object.destroy()
```

```
    def _show_table(self):
815
        #count = 1
        while 'disabled' in self._parent._button_start.config('state'):
            if (self._parent._set_var ==1 ):
                break
820         measurements = self._parent._parent._measure_obj.db_object.get_measurements()

            if measurements != []:
                if (self._parent._set_var ==1 ):
                    break
825
                self._parent._all_measurements = self._parent._all_measurements + measurements

                if self._dead_graph == 1:
                    break
830             else:

                    self.__plot_object.config(self._parent._all_measurements[0:],self)  ## be careful without [0:] it
                        is a pointer

            time.sleep(1)
835
        try:
            self.__plot_object.destroy()
        except:
            "Graph already dead"
840     print "dynamic graph ends"
            #print self._dead_graph
```

B.9 tooltip.py

```
#!/usr/bin/env python

""" create tooltips """

5

__author__ = "Carsten Koebernick <c.koebernick@rdg.ac.uk>"
__date__ = "17.10.2005"
__version__ = "0.1"
__revision__ = "1.0"
10

import Tkinter

class ToolTip(Tkinter.Toplevel):
15      """ Class creates tooltips for widgets """

    def __init__(self,text,x,y):
        """ constructor sets text, x, y -value """
        self.__text =text
20      self.__x_value = x
        self.__y_value = y
    def show(self):
        """ show the tooltip """
        Tkinter.Toplevel.__init__(self)
25      self.overrideredirect(1)
        self.geometry("+%d+%d" % (self.__x_value-15,self.__y_value+5))
        self.label = Tkinter.Label(self, text = self.__text, fg = "#000000", bg = "#ffffaa",
                                   bd = 2, relief = "raised")

30      self.label.pack(fill = "both", expand = "yes")
```

C Bash Scripts

C.1 average.sh

```
#!/bin/bash

#******* measure the average cpu_load **************#
#variable=`cat ~/.srb/.MdasEnv`
if ! [ $1 ]
then
    exit -1;
else
    var=$1
fi
ppid=$(lsof -c srbMaster | grep -i $var | awk '{print $2}' )
#| sed -e 's/*://')
if ! [ $ppid ]
then
    echo "0"
    echo "no srbMaster found"
    exit 0
fi
list=(`ps -o pcpu --ppid $ppid`)
sum=0.0
list_length=${#list[@]}
for ((i=$list_length -1;i>0;i--))
do
    sum=`echo ${list[${i}]}+$sum | bc`
done
echo $sum
exit 0
```

C.2 average_mem.sh

```
#!/bin/bash

#******* measure the average cpu_load **************#
#variable=`cat ~/.srb/.MdasEnv`
if ! [ $1 ]
then
    exit -1
else
    var=$1
fi
# get the ppid of the srbserver which is the pid of the srbMaster process
ppid=$(lsof -c srbMaster | grep -i $var | awk '{print $2}' )
#| sed -e 's/*://')
if ! [ $ppid ]
```

```
then
    echo "0"
    echo "no srbMaster found"
    exit 0
fi
list=(`ps -o pmem --ppid $ppid`)
sum=0.0
list_length=${#list[@]}
for ((i=$list_length-1;i>0;i--))
do
    sum=`echo ${list[${i}]}+$sum | bc`
done
echo $sum
exit 0
```

C.3 num_fd.sh

```
#!/bin/bash

#******* measure the average cpu_load **************#
if ! [ $1 ]
then
    exit -1
else
    var=$1
fi
ppid=$(lsof -c srbMaster | grep -i $var | awk '{print $2}' )
#sed -e 's/*://')
if ! [ $ppid ]
then
    echo "0"
    echo "no srbMaster found"
    exit 0
fi
#list srb servers with srbMaster as ppid and kill the table head and the zombies
list=(`ps -o pid,command,ppid --ppid $ppid | grep -v -i -e "\(PID\|defunct\)"`)
sum=0
list_length=${#list[@]}
for ((i=0;i<$list_length;i=i+3))
do
    # add all fd together
    let sum=`ls -l /proc/${list[${i}]}/fd | wc -l`-1+$sum
done
echo $sum
exit 0
```