Investigation and Development of Monitoring Tools 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

Andrea Weise

4th March 2006

University Supervisor: Prof. Vassil Alexandrov (University of Reading)
Dipl.-Ing. Peter Puschmann (FHTW Berlin)
Placement Supervisor: Dr. Adil Hasan
(CCLRC Rutherford Appleton Laboratory)

Acknowledgements

"Hi Andrea, You ask some interesting questions!! I am just as mystified as you are with the SRB log files."

(Roger Downing, eScience Systems Administrator)

Any project requires directions and assistance in one or other way. Although not every question got answered I would not have been able to manage this project without any help. Therefore, I would like to thank

- Dr. Adil Hasan for his supervision and support
- Dipl.-Ing. Peter Puschmann for his supervision
- Prof. Vassil Alexandrov for his supervision
- Roger Downing for trying to solve the SRB log file mystery with me
- Dipl.-Ing./MSc Marc Bartels for his critical opinion on the dissertation
- Dipl.-Ing./MSc Martin Ostermayer for his critical opinion on the dissertation
- MSc Nicholas Laurance Carter for his English corrections
- Falk Wilamowski for endless discussions about how and why and keeping me happy

For the unconditional support in so many ways especially over the last four years and for believing in my capabilities I would like to thank my parents Heidemarie and Bernhard Weise, without them my studies would not have been possible.

Abstract

The Storage Resource Broker (SRB) is a data grid management system developed by the San Diego Supercomputer Center (SDSC). The software is able to unite and manage storage media of many kinds on heterogeneous systems across the network and, as a result, to make the storage infrastructure appear transparent for the end user.

This dissertation presents the development of multiple independent monitoring tools, which operate within a network to support and improve the administration and debug process of the SRB system. Emphasis is put on the design and implementation of a software package used to successfully analyse, transfer and display the contents of the SRB systems log files.

This report discusses basic fundamentals about network communication techniques and examines state-of-the-art parsing methods. The design of the novel applications is based on a client-server-architecture. The main approach is to provide a server, which evaluates the SRB server log file and a client, which processes the parsed results of the server. Communication between the two modules is implemented via/with remote procedure calls in conjunction with the Extensible Markup Language (XML). Special attention is payed on network security though integration of encryption algorithms. To complete the set of tools and to provide more flexibility, a module to administrate the server application has been developed along with a software component to present the parsing results in a perspicuous way. Summarised, the dissertation provides inside knowledge about design and implementation issues, faced problems during the development and the corresponding solutions.

Contents

List of Figures

List of Tables

Listings

Abbreviations

ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
ANSI	**A**merican **N**ational **S**tandards **I**nstitute
awk	Alfred V. **A**ho, Peter J. **W**einberger, Brian W. **K**ernighan
bash	**b**ourne **a**gain **sh**ell
CCLRC	**C**ouncil for the **C**entral **L**aboratory of the **R**esearch **C**ouncils
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
CPU	**C**entral **P**rocessing **U**nit
DTD	**D**ocument **T**ype **D**efinition
DOM	**D**ocument **O**bject **M**odel
DNS	**D**omain **N**ame **S**erver
egrep	**e**xtended **g**lobal **r**egular **e**xpression **p**rinter
ERM	**E**ntity **R**elationship **M**odel
GCC	**G**NU **C**ompiler **C**ollection
GNU	**G**NU's **N**ot **U**nix
GUI	**G**raphical **U**ser **I**nterface
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
IDL	**I**nterface **D**efinition **L**anguage
IEC	**I**nternational **E**lectrotechnical **C**ommission
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IP	**I**nternet **P**rotocol
IPX	**I**nternetwork **P**acket e**X**change
ISO	**I**nternational **S**tandards **O**rganisation
MCAT	**M**eta Data **Cat**alog
MD5	**M**essage **D**igest **5**
OOP	**O**bject **O**riented **P**rogramming
ORB	**O**bject **R**equest **B**roker
OSI Reference Model	**O**pen **S**ystems **I**nterconnection Reference Model
RFC	**R**equest **F**or **C**omments
RMI	**R**emote **M**ethod **I**nvocation
RPC	**R**emote **P**rocedure **C**all
SAX	**S**imple **A**PI for **X**ML

SDSC	**S**an **D**iego **S**upercomputer **C**enter
SHA1	**S**ecure **H**ash **A**lgorithm **1**
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
SQL	**S**tructured **Q**uery **L**anguage
SRB	**S**torage **R**esource **B**roker
SSL	**S**ecure **S**ockets **L**ayer
XDR	**E**xternal **D**ata **R**epresentation
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
UID	**U**ser **Id**entification (Number)
W3C	**W**orld **W**ide **W**eb **C**onsortium
XML	E**x**tensible **M**arkup **L**anguage

1 Introduction

Nowadays data management is an important issue, especially for companies or institutes which have to handle millions of files and tera-bytes of data. To unite different storage media in different location is often a big problem.

Modern grid technologies unite many systems within are virtual network. By doing so, computational power can be achieved which can be higher than today's super computers with relatively low costs. Grid architectures can be classified into computing grids, access to distributed computing resources, and data grids, access to distributed databases [1]. Data grids are often used to handle massive storage resources and to provide a constant availability of data. A pivotal role within data grids falls to the data management. It can be very difficult for the end user to locate the wanted data. Data migration and data replication are essential issues as well.

The Storage Resource Broker (SRB) developed as a project at the San Diego Supercomputer Center (SDSC) is such a data management application for a data grid environment and offers solutions for the aforementioned problems. The SRB system is a standalone application and relatively compact. The usage and administration of the systems showed that the analysis of the SRB system behaviour sometimes can be challenging. To ease this process this project was carried out.

The project is concerned with the investigation and development of tools that will aid in the administration of the SRB. The project provides highly configurable tools to monitor SRB server log files on remote machines.

1.1 About This Dissertation

In this section a brief overview about the dissertation structure and used conventions are given.

In **chapter 2** the background to understand the project is provided. Basic technologies are described and explained.

Chapter 3 is concerned with the analysis of existing technologies and project relevant issues. Among other things the SRB log file is analysed there as well as existing software products.

Possible solutions and specifications are presented in **chapter 4**.

Based on the made decisions a few interesting implementation aspects are surveyed more closely in **chapter 5**.

Gained results and made tests are illustrated in **chapter 6**.

Chapter 7 finally presents the conclusion about this project.

An outlook in the future can be found in **chapter 8**.

Following typographic conventions are used to make this thesis more readable:

italic	citations
bold	important statements
`typwriter`	source code or commands
[number]	reference number

1.2 Motivation

The Council for the Central Laboratory of the Research Councils (CCLRC), which supports this thesis, was founded in 1995. The CCLRC owns and operates the Rutherford Appleton Laboratory in Oxfordshire (RAL), the Daresbury Laboratory in Cheshire and the Chilbolton Observatory in Hampshire [2]. The laboratories support and drive forward research in many areas. *"New Science through the Grid."* [3] is the vision of the e-Science Centre, which is just one programme of the CCLRC. The e-Science Centre is running several different programmes, all connected to grid technology. One of the programmes is called "Data Storage and Management" and investigates the question of storing data under several aspects, like the improvement in the quality of data curation and digital preservation [3]. The Storage Resource Broker is on of the projects there.

For the development and usage of existing SRB systems it is difficult to debug or supervise a running system. Errors *e.g.* due to problems with a server connecting to a remote SRB master or if the meta data catalog (MCAT) is server down as well as data or hostConfig file errors have to be detected effectively to provide a good service. Hence, it is also not easy to evaluate the performance of the SRB system within a reasonable time. Therefore, there is a great demand for monitoring and maintenance tools for supporting the analysis and administrative work which will improve the SRB system performance and availability.

1.3 Project Description

The project deals with the investigation and development of tools for monitoring and administrating the SRB system. The main emphasis of the project are the log files of the SRB system, which have to be analysed. According to the results of the analysis a tool has to be developed, which is able to

- parse the SRB log file
- adopt individual configuration concerning
  - parsing itself (*e.g.* parsing keywords)

  - file organisation

- preprocess the parsed data
- offer an interface to access the preprocessed data
- offer an interface to manipulate the parsing process remotely

Furthermore, a tool is required which is finally processing the parsed data. Processing in this case means inserting the preprocessed data into a database, displaying the data in a clear way to the user and notifying the user via email. The tools can be divided into several parts or individual applications, but the main emphasis lies on a client-server-application, whereas the server parses the SRB log file. The client is concerned with storing the data in a database and notifying specified user via email.

The access to the application which is running on the same system as the SRB server should be secured in that way that the connection is encrypted. This is needed to secure the SRB system.

The application is to be written in the script language Python Version 2.2.3 for a UNIX operation system using an object oriented approach. Python offers many different packages, but for this application the attempt is to use the standard library and to employ as few of those additional packages as possible to keep the tools flexible and small. All applications written during this projects are individual software products and primarily console applications. To ease the evaluation of the parsing results graphs with a small graphical user interface have to be developed.

1.4 State-of-the-Art

Nowadays, data grids are becoming more and more important, especially in the academic world. Grid computing denotes all methods, which unite the computing power of all computer system within a network. A data grid offers data resources additionally. By using data grids, it is possible to use data which might be distributed on several computer systems. The grid can be designed that way that the user does not know, where exactly the data is located. This possible transparency is called data virtualisation.

As an project of the San Diego Supercomputer Center (SDSC) the Storage Resource Broker got developed.

The SDSC Storage Resource Broker (SRB) is client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. [4]

The system offers possibilities for

- collection-building
- managing data
- querying data
- accessing data
- preserving data

in a distributed data grid network. The software is used to support data grids, digital libraries, and persistent archives [5] and is running successfully in many projects.

Each SRB server is managing and brokering storage resources which can be accessed via a computing system. The SRB can be described as a federated server system. This way of implementation provides several benefits:

- Location transparency

  The user does not need to know the exact location of the data that needs to be accessed. He can authenticate at any SRB server to access any in the system stored data.

- Improved reliability and availability

  The SRB system has a certain intelligence to organise and control the stored data within itself. This may include data replication.

- Logistical and administrative reasons

  The SRB system can be run on many operation systems. Different security protocols and policies might be involved. Therefore, a single sign-on environment and Access Control Lists are maintained for each digital entity.

- Fault tolerance

  If data is not available, the system is redirecting automatically the user to the replicated data on a different system.

- Integrated data access

  Access to back-up data is Possible.

- Persistence

  Recursive directory movement enables copying data to a new system and therefore data migration, without affecting access.

Figure 1.1 gives a brief overview of the SRB architecture.

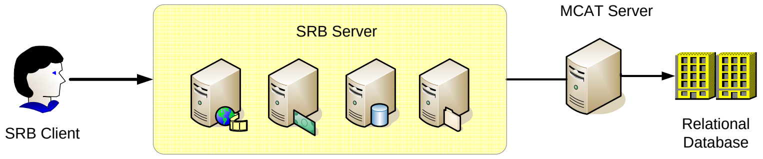


FIGURE 1.1: SRB Architecture

The core is the SRB server, which accepts enquiries from the SRB client. The server knows all meta data about users, resources and datasets. Meta data are basically data about data. They describe the stored data concerning content, type, location et cetera. This structure enables the user to gain quickly a basic summarised knowledge about data he might be interested in. Through the meta data structure, the original data is well organised and fast searchable.

Basically there are two different kinds of SRB server exist:

- SRB server with connection to a Meta Data Catalog (MCAT)

- SRB server with an MCAT

The SRB server with an MCAT or connection to the MCAT is able to authenticate clients. This is done by the master process. After the client is successfully authenticated, the master process hands the connection over to the SRB agent, who handles all the client enquiries.

Each server maintains one log file. All running SRB server processes write this log file if an event happens. This project will evaluate this log file since the SRB system is not providing such possibility. With those evaluation tools the project will support and improve the SRB system administration and the debug process and finally improve the SRB system performance.

2 Fundamentals

The SRB system can be distributed over several individual systems connected through a network. Therefore, the application parsing the SRB log file as well as the application processing the parsing results have to operate across a network.

In this chapter basic technologies in conjunction with communication techniques across networks as well as Internet security issues are explained to be able to understand the application development.

The collected data by the parser has to be structured to support quick processing. XML provides such a structure. This chapter is also used to point out XML handling and certain XML restrictions. Furthermore, the for this project used script language Python is introduced.

2.1 Basic Network Principles

A network can be described as a pool of different and individual electronically systems (*e.g.* computer systems) which are connected with each other. There are several network structures with different topologies possible such as ring, tree or bus topologies. Even a combination of different topologies are not unusual. The network enables the communication of the technical systems with each other. According to the way the data is transferred, the network can be classified as wired networks (*e.g.* Ethernet or Token Ring) or wireless networks (*e.g.* Bluetooth or networks of the type IEEE 802.11 (Institute of Electrical and Electronics Engineers)). The communication is carried out with protocols. The design of the protocols and the principles of network communication are based on the Open Systems Interconnection Reference Model (OSI Reference Model). The OSI Reference Model counts as a standard model for communication within a network and consist of seven layers. Each layer of the OSI model represents a function performed when data is transferred between cooperating applications across an intervening network [6]. A layer can contain several protocols to fulfil its requirements.

2.1.1 TCP/IP

A network protocol describes rules of data exchange, which have to be applied in order to enable communication between technical systems. These rules consist of a certain syntax and semantic to define the protocol. In our virtual world, several of such protocols exists. For example Novell introduced a network protocol call Internetwork Packet eXchange (IPX). Apple Talk was developed 1980 by "Apple Computer" to create a simple access to shared resources such as files or printers [7]. But the most common used and therefore most widely spread protocol is the Transmission Control Protocol (TCP). Together with the Internet Protocol (IP), the protocol suite TCP/IP is formed. Literature provides TCP/IP architectures with three to five functional levels. Figure 2.1 [6] shows the TCP/IP protocol architecture based on the model that the United States Department of Defence (DoD) original developed.

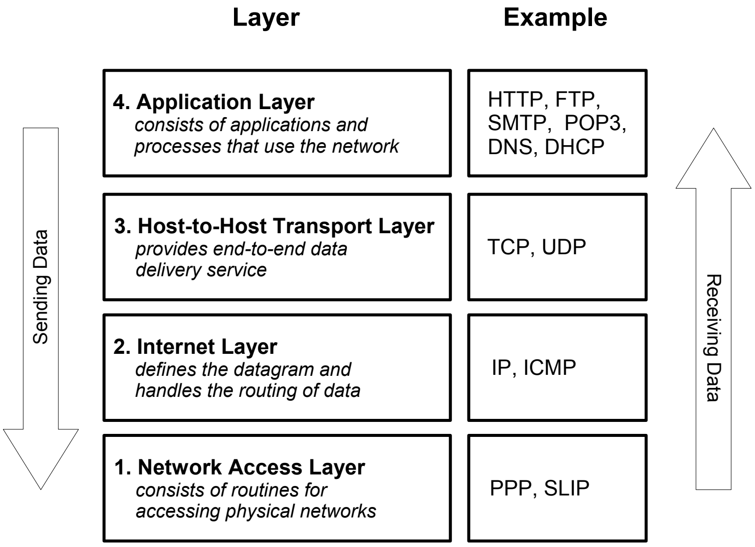

Layer
Example
4. Application Layer
consists of applications and processes that use the network
HTTP, FTP, SMTP, POP3, DNS, DHCP
3. Host-to-Host Transport Layer
provides end-to-end data delivery service
TCP, UDP
2. Internet Layer
defines the datagram and handles the routing of data
IP, ICMP
1. Network Access Layer
consists of routines for accessing physical networks
PPP, SLIP
Sending Data
Receiving Data

FIGURE 2.1: TCP/IP Protocol Stack based on the DoD-Model

Each layer provides its own structure and conventions.If data has to be send it is send down the stack to the network and vice versa if data is received. To enable compatibility and successful transmission each layer adds certain control information. This process is called encapsulation. On the receiver side each layer removes its own control information before passing on the data to the next layer above. The idea is that each layer can work without knowing the structure of the surrounding layers. In reality the layers are defined in that way, that data is passed through the stack efficiently.

The Transmission Control Protocol was standardised in 1981 under the Request For Comments (RFC) 793 by the Internet Engineering Task Force (IETF). The IETF is an international association of network technicians, producers and users, which are responsible for proposals concerning the standardisation of the Internet. TCP is situated in the transport layer of the OSI Reference Model and in the host-to-host transport layer of the DoD-Model.

To establish a connection the three way (or three message) handshake is used. The system, which is initiating the handshake sends a synchronisation packet (SYN) with a arbitrarily chosen sequence number x to the opposite system. The opposite system acknowledges the receiving by incrementing the sequence number ($ACK = x + 1$). Further, a SYN packet with another sequence number y is send back to the initiating system. Again the receiving of this packet gets acknowledged by incrementing the just received sequence number ($ACK = y + 1$). The connection is established. Figure 2.2 illustrates the procedure. Closing of the connection works similarly controlled and is shown in Figure 2.3. Instead of a SYN packet a end packet (FIN) is sent. The receiving of the packet is again acknowledged (ACK).

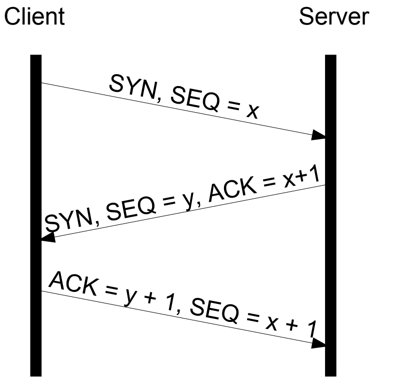


FIGURE 2.2: TCP Handshake

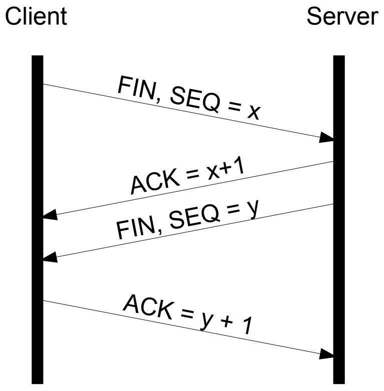


FIGURE 2.3: TCP Connection Termination

TCP is a connection-oriented and end-to-end protocol. It verifies the data integrity through a check sum in the packet header. The correct order of the packets is ensured by a sequence number. The sender sends packets again, if no acknowledge was received or a timeout occurs. The receiver is able to put the packets in the right order and discard double packets. Therefore, TCP can be considered as a reliable transfer protocol.

The Internet Protocol was standardised 1981 under RFC 791 by the IETF and the most commonly used version is the Internet Protocol Version 4 (IPv4), although IP Version 6 (IPv6) is slowly supported by more and more systems. Since TCP is organising the packets, IP is just taking care of sending the packets. Hence, IP is a connectionless protocol. There is not continuing end-to-end communication. IP can be integrated in the OSI Reference Model in the network layer and in the TCP/IP architecture in the Internet layer.

2.1.2 HTTP

HTTP stands for Hypertext Transfer Protocol. It is stateless protocol [8] for transferring data within a network and can be placed in the application layer of the OSI Reference Model and TCP/IP stack architecture. HTTP is used to transfer websites from a remote computer system to a local system. If a link like `http://www.fhtw-berlin.de/info.html` is activated, a request is sent to the system with the name `www.fhtw-berlin.de` to deliver the file `info.html`. The system name is translated to a IP address by the Domain Name Service (DNS) protocol. For the transfer the TCP protocol is used on the standard port 80. The current version is HTTP 1.1.

2.1.3 SMTP

The Simple Mail Transfer Protocol (SMTP) is defined in RFC 2821 and is used to transfer mails. Like HTTP it can be situated at the application layer of the OSI Reference Model and TCP/IP architecture. The mail, subject to a certain syntax, is send to an SMTP client. The client determines the SMTP server using other existing technologies. The mail is then send to the server directly or through other intermediary systems. The commands exchanged between client and server or the systems in between are defined in the Simple Mail Transfer Protocol.

2.2 Client-Server-Architecture

Transferring data means communication between two systems. The division of the work between the system can be derived form the host architecture. A host is a system within a network which offers services. Beside peer-to-peer architectures or mainframe architectures often client-server-architectures are found.

The client-server-architecture can be called as an architecture of distributed intelligence and is cross-platform compatible. It is possible to run client and server on different operating systems. As in peer-to-peer networks, where the load is equally distributed, the work load between client and server is divided differently. The server usually provides services, which can be resources or possibilities to access those resources (“Back End”). The client forms the “Front End” as an application to use the services the server offers.

This architecture has the advantage, that all resources are gathered and centralised at one dedicated server and they are available for many clients. The idea is to source out processing intensive tasks to the server. The client only represents the interface to the server/ processing results. The performance of the architecture depends on the server. However, if the server fails the architecture/ application fails which is a drawback of this system.

2.3 Network Security

To establish a secure connection between client and server is one of the issues in this project. But what does it exactly mean, having a “secure” connection?

First of all, what is meant by secure connection is data confidentiality. Nobody should be able to eavesdrop the sent data. Another important point is data integrity. If somebody is tampering around with the data, this should be detected by the system. Authentication is an essential issue as well. Only certain people should be able to access and operate the server.

To ensure data confidentiality cryptographic algorithms can be used. At the moment, there are quite a few algorithms, for example

- Symmetric Key Encryption

- Public Key Encryption
- Cryptographic Hash Functions
- Message Authentication Codes
- Digital Signatures

Symmetric key algorithms using only one key to encrypt and decrypt data, but once the key is discovered, confidentiality cannot be guaranteed.

Public key cryptography uses two keys, a public key to encrypt the data and a private key to decrypt. The public key gets freely distributed and everybody is able to encrypt, but only the receiver, who owns the private key is able to decrypt the message.

Cryptographic hash functions are special checksum algorithms, which produce a fixed-size output (message digest). Those algorithms like MD5 (Message Digest 5) or SHA1 (Secure Hash Algorithm 1) are meant to be one way encryption functions. They are often used for password purposes, because the same input creates always the same output. If a secret key is combined with the producing of the message digest, then those structures are called Message Authentication Codes.

Digital signatures are used to authenticate messages without the need of secret keys.

Data integrity can be detected with checksums. Authentication can be realised through passwords or certificates. A certificate is a piece of data that includes a public key associated with the server and other interesting informations, such as the owner of the certificate, its expiration date, and the fully qualified domain name associated with the server [9].

Cryptographic can provide solutions to data confidentiality, data integrity, authentication, and non-repudiation. To implement all of these features itself can be very difficult and would consume a thesis of itself. Fortunately there exist a few security suites, which are trying to implement all those ideas and still make it possible for other people to use it in a comfortable way.

2.3.1 SSL/ TSL

2.3.1.1 Overview

The most widely spread security protocol is currently Secure Sockets Layer (SSL) protocol. Developed originally by Netscape, it is designed to use TCP as a communication layer. SSL provides a reliable end-to-end secure and authenticated connection between two points over a network [10] and addresses following targets:

- Authentication

  Key cryptographic technologies, already describes on page 12 are supported to authenticate both sides within the network communication.

- Data Integrity

  The SSL protocol ensures that nobody is able to tamper with the data.

- Data Privacy

  The data produced by the SSL protocol itself and the data of the application are secured by the protocol.

To gain the aforementioned aims the SSL protocol consists of several protocols as illustrated in Figure 2.4 [10].

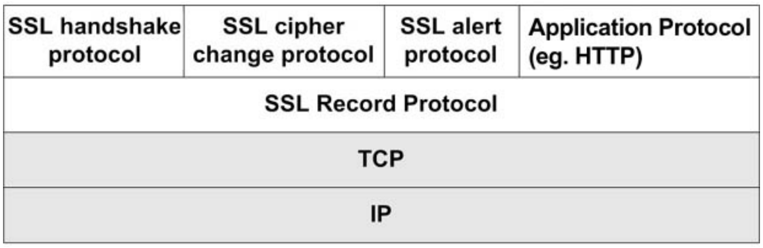


FIGURE 2.4: SSL Protocol Stack

The Application Data Protocol is responsible for the data transfer between the application and SSL. To establish a SSL connection, the SSL Handshake Protocol, the SSL

ChangeCipher SpecProtocol and the SSL Alert Protocol are used. These three protocols cover the areas of session management, cryptographic parameter management and transfer of SSL messages between the client and the server [10]. The Alert Protocol is used to forward warnings and error messages. The ChangeCipher SpecProtocol initialises the cryptographic procedure. Through the Handshake Protocol, server and client negotiate the cryptographic procedure. Data encryption, data integrity, and if required data compression is assured by the SSL Record Protocol. This protocol is also able to encapsulate data, which is sent by other SSL protocols and is therefore responsible for the SSL data check.

2.3.1.2 SSL Handshake

The SSL handshake is the basis for each SSL connection and has a particular importance. Figure 2.5 [10] shows a possible SSL handshake for establishing a connection.

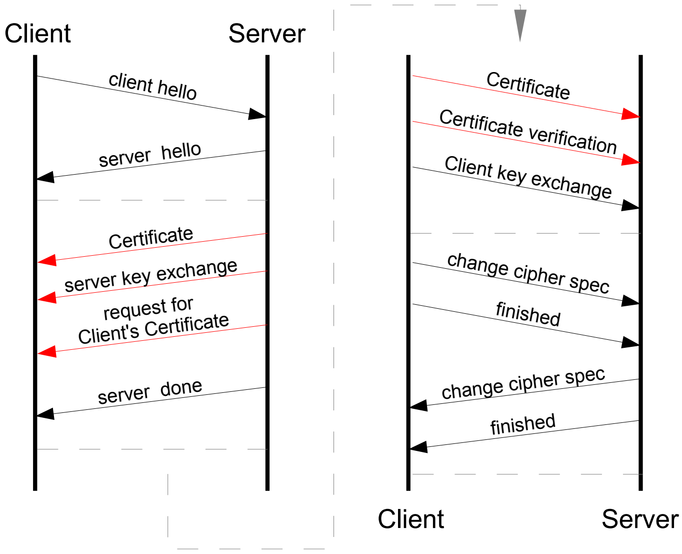


FIGURE 2.5: Possible SSL Handshake (red = optional)

The client starts the connexion establishment by sending a "client_hello", a so-called challenge (value), a list of supported cryptographic and compression procedures, and if available, a session identification from an earlier session to the server.

The server chooses a procedure and answers with a "`server_hello`". If the indicated session identification is found in the servers's cache, the former agreed master key can be used. Otherwise the server sends his certificate (optional, needs to be requested from the client), which can be one or a chain of certificates, and the chosen codes (cryptographic and compression procedures) to the client. Depending on the negotiated method of key exchange, the server sends a ServerKeyExchange message which is a list of certificate types. The server finishes his part by requesting a client certificate (optional) and sends the "ServerDone" message.

The client generates a master key and encrypts this key using the servers's public key. The encrypted master key is sent back to the server. The master key and connection concerned data is used to derive a session key by using a hash function (*e.g.* MD5). The session key is required for the data encryption. The master key is not used for that. For each direction (sending and receiving) an individual session key is generated. Finally, the client encrypts the connection identification with its own session key and sends it to the server including the finished message. The server encrypts the challenge with his session key and sends it to the client including his finished message. The client verifies if the challenge has the same value as the challenge he has sent at the beginning. If the values are identical, the client knows that the servers certificate is authentic. Otherwise the server would not have been able to decrypt the master key.

The server has the possibility to verify the authenticity of the client, too. The request contains a challenge value and a list of available authentication procedures. The client responds with his certificate and authentication information.

After the handshake completion the data gets encrypted according to the agreed procedure. A Message Authentication Code is added to the data to ensure data integrity.

2.3.1.3 Remarks

The current version is SSLv3. Version 2 is still available but is considered as insecure, because of fundamental design problems [9] and should **not** be used.

In 1996, the IETF standardised Internet security methods and they used SSL 3.0 as a basis. Under RFC 2246, they released 1999 a new Transport Layer Security (TLS) protocol version 1.0. TSL implements the same features as SSL and contains additionally more interoperability and expandability towards applications. Additional RFC's and extensions have been published by the IETF in connection to TSL. At the moment, the IETF works on the new version TLS 1.1, a draft is available so far.

TLS can be seen as the successor of SSL and often both terms are used equally.

2.4 XML

This project requires a platform independent possibility to exchange data. A very flexible way of data exchange offers the Extensible Markup Language (XML). XML is the state-of-the-art in that area and there are hardly any other technologies which provide such flexibility. Using XML for this project provides also an interface for any other application to use the gathered data.

2.4.1 Overview

XML is a subset the Standard Generalized Markup Language (SGML) defined by the International Organisation for Standardisation (ISO) 8879.

XML is a markup language for documents containing structured information [11]. A markup language is a mechanism to identify structures in a document [11]. Defined by World Wide Web Consortium (W3C), XML describes rules for the document layout. A simple XML document as an example is listed in Listing 2.1.

LISTING 2.1: XML File Example

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE message [
 <!ELEMENT message (entry)>
 <!-- a message consists of entries -->
 <!ELEMENT entry (date, time, error_number, error_string, linenumber)>
 <!-- entry contains  date, time, error_number, error_string, linenumber-->
    <!ATTLIST entry
      number  CDATA #IMPLIED
    >
  <!ELEMENT date (#PCDATA)>
  <!-- data contains the data text and nothing else -->
        <!ATTLIST date
          typ  CDATA #REQUIRED
        >
  <!ELEMENT time (#PCDATA)>
  <!-- time contains the time text and nothing else -->
17     <!ELEMENT error_number (#PCDATA)>
18     <!-- error_number contains the error_number text and nothing else -->
19     <!ELEMENT error_string (#PCDATA)>
20     <!-- error_string contains the error_string text and nothing else -->
21     <!ELEMENT linenumber (#PCDATA)>
22     <!-- linenumber contains the linenumber text and nothing else -->
23 ]>
24
25 <message>
26 <entry number="1">
27 <date typ="database">2005-10-23</date>
28 <time>01:00:01</time>
29 <error_number>-</error_number>
30 <error_string>portalConnect: connect msg timed out for pid 25133</error_string>
31 <linenumber>10280</linenumber>
32 </entry>
33 </message>
```

In the first line the XML declaration is found. This declaration consist of a “<” followed by a “?” and the word “`xml`” in small letters inclusive the closing “>”. Here the used XML “`version`” can be defined, too. The current version is 1.0 which is supported by most common parsers. The optional attribute `encoding` defines, which character encoding is used for saving the XML file. With the noncompulsory attribute `standalone` it is possible to tell the parser if the file refers to an internal or external Document Type Definition (DTD), where as `standalone="yes"` indicates an internal DTD.

The Document Type Definition describes the possible elements, attributes, entities and nesting possibilities of a XML document. The DTD seperates the data from the data definition. DTD are used to validate the XML document. In the given example, an internal DTD defines the rules (lines 2 - 23). The document type declaration starts with `<!DOCTYPE` followed by space and the name of the document type. Then the `ELEMENT message` is introduced. `message` consist of the element `entry`, where `entry` again is formed of the elements `date`, `time`, `error_number`, `error_string` and `linenumber`. Elements can have attributes, indicated by the keyword `ATTLIST`. The keyword `#REQUIRED` defines, that the attribute has to have a value, the opposite is indicated by the keyword `#IMPLIED`. The elements `date`, `time`, `error_number`, `error_string` and `linenumber` finally carry the data. A `ENTITY` defines a “wildcard”, which can be used later in the document. Names for elements, attributes and entities can consist of

- letters (capital and small),
- numbers (0 till 9),
- punctuation characters like
  - _ (underscore),
  - - (hyphen),
  - . (dot),
  - : (colon), where as the colon is reserved for namespaces.

The first character has to be a letter or any allowed punctuation character. Names must not have a space.

The actual XML file (lines 25 - 33) has to use exactly the same elements defined in the DTD above. Each element is framed by a start tag (<element name>) **and** an end tag (</element name>). If there is a syntax error, the XML document is not "well-formed". From the rules defined in the DTD it is clear, that the elements

- <date> ... </date>
- <time> ... </time>
- <error_number> ... </error_number>
- <error_string> ... </error_string>
- <linenumber> ... </linenumber>

can only be within the element <entry> ... </entry>. A value assignments have to be in quotes. Everything between $<!--$ and $-->$ are comments and are ignored by the parser.

2.4.2 Restrictions

An XML file is considered as "well-formed" and therefore abides the rules, if

- the file has an XML declaration which refers to XML

- there is always a start and end tag
- there is at least one data element
- there is a element that contains the data
- all attribute values wrapped in quotes
- all attributes do not contain the character “<”

An XML file is “valid” if the rules defined in the DTD are implemented. Thus, “well-formed” and “valid” are different subjects concerning XML files. The design and use of a DTD is not mandatory and in many cases not necessary, for example if the parser is not verifying the validity of the document.

Within an XML file, all characters of the norm ISO/IEC (International Electrotechnical Commission) 10646 (unicode system) are allowed:

- hexadecimal values #x20 to #xD7FF
- hexadecimal values #xE000 to #xFFFD
- hexadecimal values #x10000 to #x10FFFF
- hexadecimal values #x9 (tabulator), #xA (line feed) and #D (carriage return)

2.4.3 API’s

To extract, analyse and preprocess XML structures, a so-called parser is used. Figure 2.6 describes how a parser might work. In general there are two different kinds of parsers. First of all, the parser which validates the source code, what requires a DTD and a parser, which do not do validation.

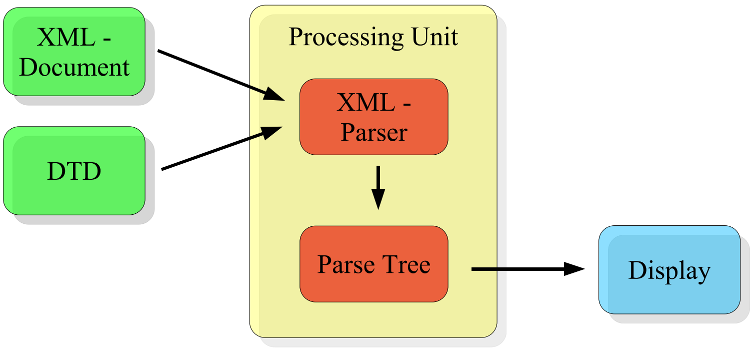


FIGURE 2.6: Possible XML Processing

The basic functions of each parser package are indicated by the “XML - Parser”. Most parsers also offer the possibility to save the document as a tree structure. The tree structure complies the Document Object Model (DOM) according to the W3C. Two of the most application programming interfaces (API) used by XML parsers are DOM and SAX (Simple API for XML).

2.4.3.1 DOM

The DOM is an application programming interface for HTML and XML documents. The architecture within the document is oriented on a tree structure. The individual nodes can be seen as objects which have functions and identities. The DOM establishes:

- interfaces and objects for representing and manipulating the document
- syntax of interfaces and objects
- connections among interfaces and objects

The data is placed into the objects where it is protected from external manipulation. DOM defines functions to manipulate the data.

2.4.3.2 SAX

The SAX API is no W3C standard and deals with the XML information as a single unidirectional stream. That means, it is not possible to manoeuvre within the document like it is possible using the DOM. If data has to be re-read the document has to be parsed again. The SAX parsers are implemented as an event-driven model.

2.5 Database

A database is an organised collection of stored data. Usually its contents can be accessed, managed and updated easily. There are several types of databases. The most common used database is a relational database which is a tabular database. A relational database consists of several tables which are connected with each other through relations. Each table contains datasets. A datasets consists of several attributes. Unique keys enable explicit mapping of datasets. A distributed database is spread of several nodes within a network. In object-oriented databases, the data is defined in object classes and subclasses.

It is assumed that the process of normalisation during a database design is common knowledge and is therefore not explained. Further explanation can be found in relevant literature such as “Introduction to Database Systems” by C.J. Date [12].

2.6 Python - “Batteries Included”

Python, named after “Monty Python’s Flying Circus”, was developed by Guido van Rossum in the Netherlands in 1990 and is recognised as a successor of the ABC programming language. Python is considered as script language.

Python is an interpreted, interactive, object oriented programming language [13]. It is a multi-paradigm language, allowing several styles of programming such as object orientated or structured programming. Data types are dynamically managed and it uses garbage collection as a memory management. Garbage collection is a method which frees regularly and automatically unused memory and other system resources. Objects which are not reachable within the memory are automatically freed.

To be simple and concise, Python consist of only a few key words and the grammatical syntax is reduced and optimised to support lucidity.

Python differs from other programming languages in terms of the code structure, as it uses the indentation itself to create blocks. Listing 2.2 and Listing 2.3 show a comparison of a function written in C and Python, respectively.

LISTING 2.2: A function in C.

```
int test (int choice, int value)
{
    if(choice == 0)
    {
        printf("nothing chosen");
        return value;
    }
    else
    {
        printf("choice: %d",
            choice);
        value += choice
        return value;
    }
}
```

LISTING 2.3: A function in Python.

```
def test (choice, value):
    if choice == 0:
        printf "nothing chosen"
        return value
    else:
        printf "choice: ", choice
        value += choice
        return value
```

All data and programming components are objects since Python is an object oriented language. However, there is no enclosing class and an object does not necessarily have to belong to a certain class. A name is bound to an object what can be very helpful. But misused with changeable object, it can cause serious side effects.

Python consists of a large standard library, which explains the “batteries included” philosophy. Modules of the standard library can be extended. The library is especially customised for Internet applications, many standards and protocols like HTTP are supported. Modules for creating interfaces to graphical components and databases are included as well as a module for regular expressions. Most of Python’s modules are platform independent and a lot of additional modules in many different areas are available.

Python is a project requirement. Nevertheless is this choice no disadvantage compared to other script languages like PERL due to the comprehensive standard library and the possibility for object orientated programming. This script language is adequate for small and large projects and is as powerful as any other script language.

3 Analysis

In this chapter it will be analysed if already existing technology can be used to fulfil the requirements. It is examined which strategy is efficient to establish network communication. Furthermore the analyses referring to the SRB log file, creating daemons within UNIX environments and security are presented. Finally a database application is introduced.

3.1 Existing Parsing Technologies

The purpose of log files is to keep track of events. Many software applications produce line after line, page after page and it seems to be a never ending stream of data. To examine those data can be difficult, not only because each log file may have a different structure. Additional knowledge may be needed to interpret the data and not all the data is important. But how to determine, which data is worth looking at?

This project demands a log file parser which is

- identifying any defined error
- dynamically configurable
- efficient to use
- accessible and manageable with Python
- running on a UNIX system
- free of use (if extra software package)

Internet research resulted that many different log parser exists. A lot of them are not freely available or written for a Microsoft Windows environment like the Microsoft Log Parser[1]. Most of the parsers are standalone applications and a special interface is needed to use it for this project. Often only a certain log file structures can be handled.

A very interesting module is the “pyparser” module for Python. The grammar can be directly implemented into the Python code. The pyparsing module is an easy-to-use Python module for constructing and executing basic text parsers [14]. The module is useful for evaluating user-definable expressions, processing custom application language commands, or extracting data from formatted reports [14]. Unfortunately, the pyparsing library requests Python Version 2.3.2 or higher, but this project is developed with Python Version 2.2.3.

Another approach is the use of parsing generators. A parser generator is a tool that creates a parser based on a certain grammar. The generated parser can also contain the source code which is executed if the defined rules apply. In the Python world a few parser generator exist such as the “Toy Parser Generator”[2] or the “Yappy”[3]. To be able to handle the parser generator a grammar to decribe the parser has to be learned. Usually this grammar is very complex, since every possible pattern can be defined. Further, additional software packages might be involved.

Instead of trying to adjust existing software solutions the decision was made to develop an own parsing module. Only one text file has to be parsed. The requirements on the parser are not that demanding and the parser could be held small and easy. This solution also does not require additional modules. The parsing could be combined with the creation of an XML file which contains the parsing results.

3.2 Communication Technologies

The communication for the required client server application is done with the network protocol suite TCP/IP since it is the state-of-the-art. The following section are possibilities to communicate through the network using TCP/IP.

[1] Microsoft Log Parser - http://www.logparser.com
[2] Toy Parser Generator - http://christophe.delord.free.fr/en/tpg
[3] Yappy - http://www.ncc.up.pt/fado/Yappy

1. Sockets

   Sockets are the basis for each communication through the network and can be described as communication end points between two programs, which are communicating through the network. Sockets are part of the operating system and can be acquired by applications. The operating system is responsible for managing the sockets.

2. Remote Procedure Call

   A Remote Procedure Call (RPC) is a mechanism which gives the possibility to execute procedures on remote systems across a network. This technique is often used in client server applications. Usually, the server provides certain procedures. The client sends a RPC request to the server and invokes the execution of this function on the server side. The server sends the return value of the procedure back to the client. Due to operation system independency the data which is exchanged between client and server gets converted. This process is called marshalling. In case of RPC the data gets converted to the External Data Representation (XDR) format by the sender. The receiver converts the data back depending on the operation system.

3. Common Object Request Broker Architecture

   The Common Object Request Broker Architecture (CORBA) is an object oriented middleware. Within CORBA are protocols and services defined which facilitate the creation of distributed applications in heterogeneous environments. CORBA is language independent and uses a Interface Definition Language (IDL) to create an interface description which is translated into the target language such as Java or C++.

   The client calls a stub code as a local connecting point. A stub is piece of code, which stands for another code which in this case is situated on another system. The stub forwards the data to a Object Request Broker (ORB). The ORB sends the data then to the ORB on the remote system. On this system a skeleton is called. A skeleton is a piece of code as well. In this case the skeleton is doing the mashalling. The stub and skeleton can be generated by an IDL compiler.

4. Remote Method Invocation

   Remote Method Invocation (RMI) is basically a proprietary Java RPC. The client calls a remote Java object. This object can be located in a virtual machine. As for RPC, the procedure calls are handled as a local procedure calls.

The requirement of using Python 2.2.3 limits the choices. Python provides a good support for RPC. There are several packages to implement RPC like the module `SimpleXMLRPCServer` which is part of the standard library. The client server communication can be implemented in a efficient way. Within the RPC package, sockets are used and the socket implementation is stable and reliable.

3.3 Daemon

Applications of this project will be running in the background. Therefore those applications should be turned into a daemon process. A daemon is a process with special characteristics. First of all, a daemon has as a parent process, the init-process, and therefore the daemon is not attached to any terminal. A daemon has super user rights and that is why the User Identification (Number) (UID) = 0.

To create a daemon in a UNIX environment certain rules and the following sequence have to be respected:

1. `fork`
   First of all, `fork` needs to be called. `fork` creates a new process whereas the initiator of `fork` is called parent. The newly created process is the child and is a copy of the parent. Parent and child have same user ID and working directory as well as the same open files.

   The parent process exits. By doing so, the terminal returns and new commands can be entered. The child process inherits the process group ID, but also gets a new process ID. The child process cannot be the process group leader.

2. `setsid`
   Calling the command `setsid` creates a new session, that leads to:

   - the process becomes session leader of the new session
   - the process becomes process group leader of the new process group
   - the process has no control terminal anymore

3. `fork`
   This second fork is executed to prevent zombies. A zombie is orphaned process table entry which occurs if a parent process is not waiting for the child to finish. Usually, the parent waits for the child's exit status, but in case the parent is not waiting, this status is kept in the process table entry. By doing the second fork, the immediate child exits. Therefore the grandchild becomes an orphan whereas the init-process emerges as responsible for the clean up of the grandchild process [15].

4. change directory
   Sometimes the process inherits a directory which needs to be unmounted. Since the daemon is still accessing the directory, unmount is not possible. Hence a directory change might be useful.

5. `umask`
   `unmask` set the file creation mask for a process. The file creation mask defines which rights are **not** to assign to new file or directory. By executing `umask` it is ensured that the child gets the correct access rights for its own files.

6. file descriptor
   Finally all inherited and open file descriptors have to be closed.

3.4 SRB Log File

The SRB system writes only one log file. This log file is accessed by various processes. The log file `srbLog`, located in the `SRBInstall/data` directory, logs all activities of the current SRB server session. If a SRB server is started, the current content of the `srbLog` gets transferred to `srbLog.sav` or in the latest version gzipped and information of the new sessions are saved again in `srbLog`. At a certain configurable interval a log file rotation is taking place. The current `srbLog` is gzipped and placed into a separate directory. The log file name is changed to include a datestamp.

In this project, the interest lies on error messages. Through the investigation of log files it seems that the SRB server errors have negative error numbers as normal system errors have positive error numbers. As for the SRB server a pattern for some log entries can be found. The example in Listing 3.1 represents the pattern of most SRB log file entries.

LISTING 3.1: SRB Log File Entry

```
NOTICE:Oct  3 20:35:04: resolveContainer: mdasGetInfo error for container testcont.
     status = -3201
```

Surveying the log file entries leads to the conclusion that in general the log entries have the following pattern:

```
<Type>: <Timestamp>: <Message>
```

where the type specifies the importance and can be

- NOTICE
- FATAL
- DEBUG
- WARN

The timestamp consists of

- **no** year but
- short version of the month name (e.g. OCT), followed by
- the day of the month as a decimal number, followed by
- the time (hour:minute:second).

The message is a short description of the event that tool place and it can contain error numbers. The SRB server system provides an error description file which contains the negative error numbers, error names and sometimes a short error descriptions.

But there are also entries in the log file, which do not follow this pattern as shown in Listung 3.2.

LISTING 3.2: SRB Log File Entries

```
getAndQueHostName: gethostbyname error for mda-18.sdsc.edu ,errno = 22
LocalHostName: zebedee.local, localhost, 130.246.42.39, 192.168.0.2, 127.0.0.1, 192.168.0.2, Port Num: 5544.
Local storage vault conf:
storSysType: 0, vaultPath: /Users/hasan/work/SRB/Vault
Local Zone :
ZoneName = AdilZ  HostName = zebedee.local  PortNum = 5544
Remote Zone :
findServerExec: found ”/Users/hasan/work/SRB/SRBInstall3.3.1/bin/./srbServer” using argv[0]
```

For those messages no reliable pattern could be assigned.

The log file size depends on the frequency of log file rotation and on the frequency of events between two rotation processes.

It was neither possible the talk to the developer of the SRB system about the creation of the SRB log file nor to acquire a relevant system description. Thus, all the results mentioned before are based on observing the SRB system and analysing existing SRB log files as well as a result of discussing the subject with people at the CCLRC.

3.5 OpenSSL

As discussed in section 2.3 implementing all the mentioned security aspects is very complex. The open source project OpenSSL is a way to utilise security features as describted in section 2.3.

OpenSSL consists of a cryptography library and an SSL toolkit and is a derived work from SSLeay which was originally written by Eric A. Young and Tim J. Hudson in 1995 [16]. In December 1998 the first version of OpenSSL was realised. Nowadays security is an important issue and the OpenSSL library is usually installed on UNIX operating systems.

The SSL library provides the user with all versions of the SSL protocol. This also includes the Version 1 of TLS. The cryptography library offers most common used algorithms which are already mentioned in section 2.3. OpenSSL is a free SSL implementation and is executable on most platforms.

As an interface to the OpenSSL library there are two Python modules available.

1. pyOpenSSL
   PyOpenSSL is a Python wrapper and the package provides a high-level interface to the functions in the OpenSSL library. It is freely available under the terms of the GNU Lesser General Public License and requires Python Version 2.1 or higher [17]. The current version is pyOpenSSL-0.6.

2. M2Crypto
   M2Crypto is a crypto and SSL toolkit for Python and the current version M2Crypto-0.13 requires Python Version 2.[1234], OpenSSL 0.9.7 and SWIG 1.3.2.[123]. SWIG is a software development tool. It is an interface compiler that connects programs written in C and C++ with scripting languages such as Perl, Python, or Ruby. It works by taking the declarations found in C/C++ header files and using them to generate the wrapper code that scripting languages need to access the underlying C/C++ code [18].

   M2Crypto consists of two layers. The lower layer uses SWIG to hook up the OpenSSL C API functions, making these available as Python functions [19]. The upper layer provides Pythonic object-oriented interfaces to the lower layer [19].

Both interfaces were investigated. For the M2Crypto module a good documentation and some examples of use were given by the developer. Furthermore, the handling was understandable and efficient. Therefore, the decision was made to use M2Crypto instead of pyOpenSSL, because the documentation was insufficient and no examples are available.

3.6 SQLite - A Light Database Engine

The project requires a database to store the parsing results. Many different types are available on the market. For this project a database is required which is freely available, runs under UNIX and is accessible by Python. Databases such as PostgreSQL, MySQL, and SQLite provides this. PostgreSQL and MySQL are complex database systems with many features. Due to the complexity both database require a certain knowledge to install and administrate the system. The opposite is SQLite. SQLite also needs less resources than PostgreSQL and MySQL due to the smaller complexity.

The parsing results contain only

- characters according to ISO/IEC 10646
- date
- timestamp

This are standard database attributes. Therefore a light database can be used. This brings performance and configuration benefits. After examining the aforementioned database engines the decision was made to use SQLite. The extensive features of PostgreSQL and MySQL are not needed for this project.

SQLite is a small C library that implements a self-contained, embeddable, zero configuration SQL database engine [20]. The transactions made are atomic, consistent, isolated, and durable [20] and no administration is required. The database is stored in a single file and it is suppose to be faster than any other common client/server database engines for most common operations [20]. Furthermore, it implements most of the SQL-92 standard. The database query language SQL (Structured Query Language) is one of the most common used query languages. To be compatible with the Python Version 2.2.3, the SQLite Version 2.8.16 is used.

To use the SQLite library an interface is needed. For this project pysqlite is used. Pysqlite is a database interface for SQLite and is freely available. Due to compatibility the version pysqlite 1.0.1 is used.

3.7 Graphical User Interface (GUI)

Although all the software, which is going to be developed, is controllable though a console this project has a small graphical aspect. The parsing results should be represented as graphs, additionally these graphs have to be savable. The graphical user interface should be self-explanatory within its handling.

The Python standard library offers the interface Tkinter to the Tk GUI toolkit and can be used for this project. TK is an open source cross-platform widget toolkit, which offers functionality for the development of a graphical user interface. The TK toolkit is usually installed on UNIX operated system.

4 Design

In this chapter all design issues concerning the software development are presented and explained. First a few general aspects are given. These ideas apply to all applications. After that, ideas to each application are illustrated as well as class diagrams. Short explanation to all member variables and function within the classes are also given. This chapter also includes the software specifications.

4.1 General Aspects

All applications are written as a console application. That means, mainly parameters are used to control the applications. The software is designed for administrators or scientists which are using the SRB system. Consequently, basic knowledge and understanding towards handling a console application can be expected.

The software is written for a UNIX environment. To compile additional software a C-compiler is required. The GNU Compiler Collection (GCC) is most common used open source compiler and usually installed on UNIX operated systems.

According to the project description, two main applications are needed. First of all a server, which is handling the log file parsing. Second, a client which collects the parsing results from the server and is handling the storing and displaying of the parsed data as well as the notification. The design of those two applications is based on the client-server-architecture.

One server monitors one SRB system only. A client is collecting data from many servers. This relationship is clarified in Figure 4.1.

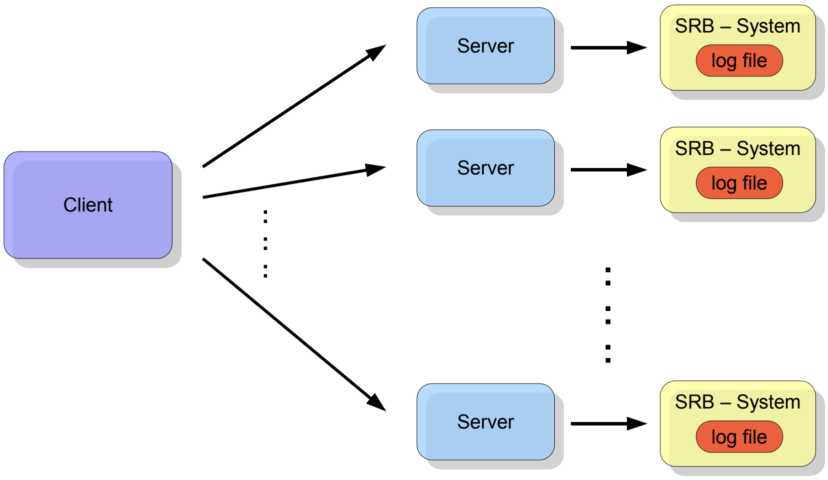


FIGURE 4.1: Client-Server-Relation (1:n)

As decided in analysis (chapter 3) the communication is done with an adjusted version of the Python standard library module `SimpleXMLRPCServer`. The handling with password is avoided due to efficiency. Therefore, authentication is done with certificates. Only SSLv3 is used. The used ports are freely configurable unless it is not a port number below 1025 and above 50000. Ports from 0 to 1024 are usually reserved for other services and an interference should be avoided.

Once the server is started, it keeps track of all log file changes after that. But to integrate older log files which are stored as compressed files (*.gz) too, a separate tool is developed. This integration is done only once, therefore this process is sourced out to another tool. The additional tool uses the same parsing technology as the server.

In the adjacent class diagrams often a `utils_xxx` class can be found. This is **not** a class, it represents a script which contains functions, which are needed by all the other classes. The class diagrams are only short versions, full versions are available in the appendix in chapter B.

4.2 Server

Figure 4.2 shows the basic client-server design approach.

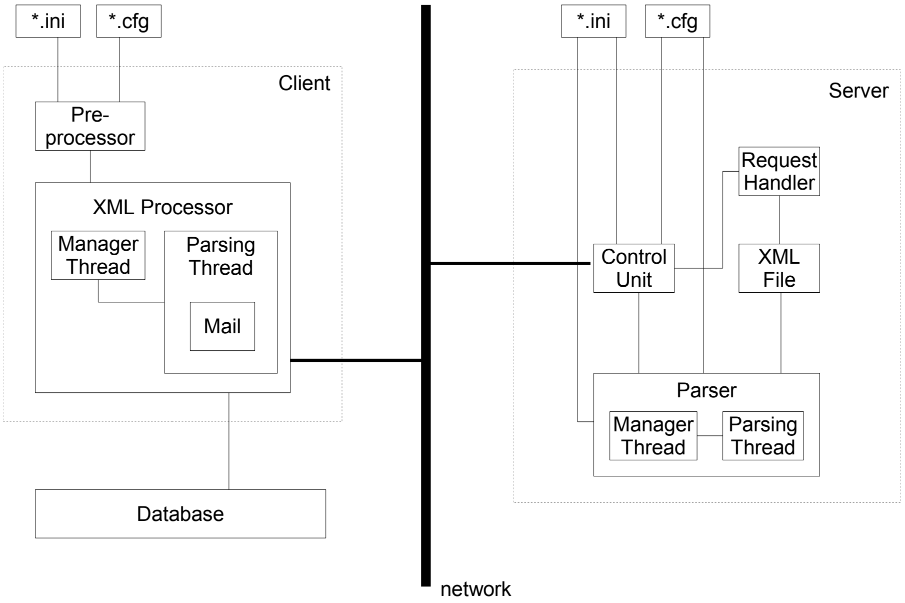


FIGURE 4.2: Basic Client-Server Design

The server's control unit is responsible for verifying the user input. Furthermore, this unit starts the parser with the right configuration and accepts incoming requests. The request are passed on the request handler. The parser works independently controlled by the manager thread. The parsing thread is doing the actual SRB log file handling. The parsing results are saved in an XML structured file, which also is accessed by the request handler.

The server has its own configuration file to gain the required flexibility. The configuration file structure is similar to Microsoft Windows INI files (*.ini). The Python standard library module `ConfigParser` is able to handle this file structure. The file structure contains section headers followed by a name including a value. Comments are applicably by using "#" or ";" characters. With the configuration file it is possible to configure

- the location of
  - server certificate file
  - certificate authority file
  - SRB log file

  - keyword file

- the parsing interval time
- the port
- the network interface (e.g. eth0)
- error numbers, which are to be ignored

The server is able to parse and handle incoming requests at the same time. This is realised with threads. By using threads it has to be ensured that several threads are not requiring the same resources at the same time. Thread synchronisation is done with mutex mechanisms. If such mechanisms are used, a system of deadlock avoidance has to be established.

The parsing module is analysing the log file by reading the SRB log file line by line. The extracted line is examined according to a keyword list. This list is defined in an additional file and the exact approach is explained in chapter 5. If the line is as wanted identified the following values are extracted:

- date
- time
- error number
- error string
- line number

The date and time is extracted from the SRB log file line. If no date or time is available, they are taken from the log file properties. The error number is compared with the given list of "ignored error" numbers. In case the number should be ignored, the parser goes on with the next line in the log file. With "error string" the whole log file entry line is referred. Line number is the actual line number in the SRB log file. The values are saved in an XML file before the parser moves on to the next line. If no error number is available, the character "-" is inserted instead.

If the parser is writing the XML file, the client has to wait until the parsing process is finished and vice versa. This is controlled with a mutex class where the same object of this class is passed on to each thread.

The communication part in the `SimpleXMLRPCServer` is exchanged to a secure server, provided by the M2Crypto package (introduced in chapter 3).

If an applications is trying to connect to the server, the request gets accepted if the SSL handshake is successfully done. The accepted connection is then passed on to a thread (`MyClientThread`). If the connected application is satisfied the thread dies automatically.

The user has the option to start the server as a daemon. The daemonisation process is implemented as described in the analysis (chapter 3). Furthermore, the user is allowed to observe the work of the server by activating the verbose mode. If the verbose mode is activated and the server is running as a daemon, the output is written into a log file which is cleared each time the server is restarted. The configuration file is handed over as a parameter, too. Table 4.1 shows the parameter for the server.

TABLE 4.1: Server Parameters

Parameter	**Explanation**
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]
-d or –daemon	daemonise the server

If the option `-h` or `--help` is used, all other given parameters are disabled.

4.2.1 Server Class Diagram Design

Figure 4.3 depicts the class diagram for the server.

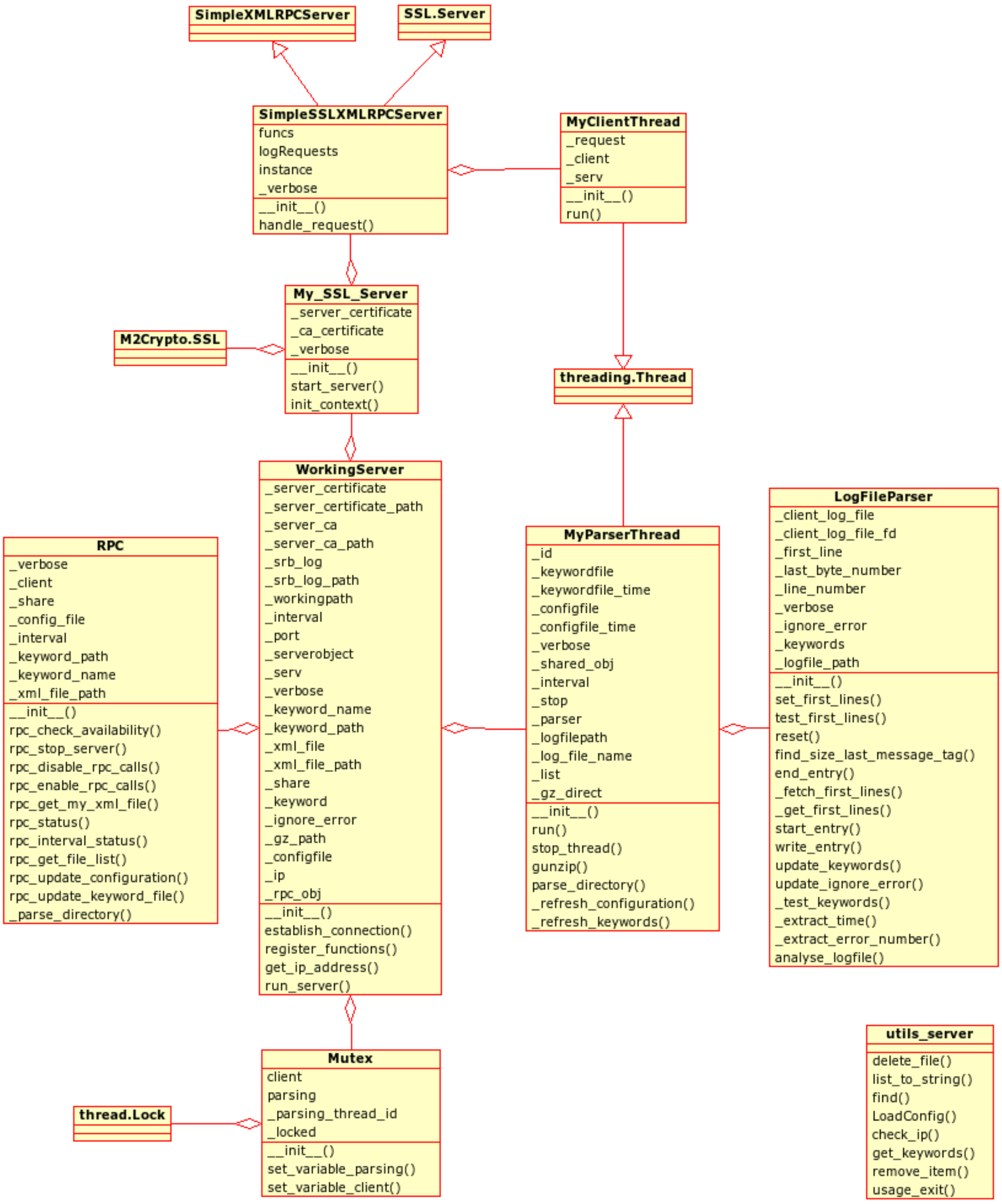


FIGURE 4.3: Server Class Diagram

The server has a manager class `WorkingServer` which consists of

- mutex object(`Mutex`)
- RPC functions (`RPC`)

- a parsing thread (`MyParserThread`)
- a secure server (`My_SSL_Server`)

and is initialising all necessary objects. The required data are hold in the member variables described in Table 4.2.

TABLE 4.2: Member Variables Class `WorkingServer`

Variable	Type	Explanation
`_ip`	STRING	IP address of network interface
`_server_certificate`	STRING	name of the server certificate
`_server_certificate_path`	STRING	location (path) of the server certificate
`_server_ca`	STRING	name of certificate authority file
`_server_ca_path`	STRING	location (path) of the certificate authority file
`_srb_log`	STRING	name of the SRB log file
`_srb_log_path`	STRING	location (path) of the SRB log file
`_workingpath`	STRING	name of working directory
`_interval`	INTEGER	parsing interval period in minutes
`_port`	INTEGER	port number
`_serverobject`	MY_SSL_Server	object of class `My_SSL_Server`
`_serv`	SIMPLESSL-XMLRPC-SERVER	object of class `SimpleSSLXMLRPCServer`
`_verbose`	INTEGER	defines printing of debug messages
`_keyword_name`	STRING	name of keyword file
`_keyword_path`	STRING	location (path) of keyword file
`_xml_file`	STRING	name of XML file
`_xml_file_path`	STRING	location (path) of XML file

Continued on next page

Table 4.2 Member Variables - *continued from previous page*

Variable	**Type**	**Explanation**
_share	MUTEX	object of class Mutex
_keyword	STRING	array of the defined keywords
_ignore_error	INTEGER	array of error numbers which are to be ignored
_gz_path	STRING	location (path) of gz files
_configfile	STRING	name of configuration file
_rpc_obj	RPC	object of class RPC

The constructor __init__ verifies the user input and initialises most of the member variables. The function establish_connection starts the server and afterwards the manager thread MyParserThread. register_function registers all RPC function to be able to use them later. With the function get_ip_address the IP address is extracted from the given network interface. Finally the function run_server accepts incoming requests.

The MyParserThread class is handling the parsing and is running as a thread. This member variables are displayed in Table 4.3.

TABLE 4.3: Member Variables Class MyParserThread

Variable	**Type**	**Explanation**
_id	INTEGER	thread identification number
_keyword_file	STRING	name of keyword file
_keyword_file_time	INTEGER	last modified time of keyword file
_configfile	STRING	name of configuration file
_configfile_time	STRING	last modified time of configuration file
_verbose	INTEGER	defines printing of debug messages
_shared_object	MUTEX	object of class Mutex

Continued on next page

Table 4.3 Member Variables - *continued from previous page*

Variable	**Type**	**Explanation**
_interval	INTEGER	parsing interval period in minutes
_stop	INTEGER	define stopping of thread
_parser	LOGFILE-PARSER	object of class LogFileParser
_log_file_name	STRING	name of the SRB log file
_logfilepath	STRING	location (path) of the SRB log file
_list	STRING	array to hold file names
_gz_direct	STRING	location (path) of the gz files

The constructor __init__ initialises the member variables. The thread can be terminated manually by using the function stop_thread. The function gunzip is used to uncompress the gzipped files. With parse_directory the newest gz files is determined. The determination is based on the last modified time taken from the file property. _refresh_configuration and _refresh_keywords are used to update the member variables which are involved in the parsing process. These function are necessary due to the possibility to change the configuration and keyword file remotely. Within run the periodically parsing is organised. First it is checked if the configuration file and keyword file were modified. In that case, the member variables get updated. Then the first lines of log file are analysed to check if a log file rotation took place. In the case of log file rotation the gz file is determined and the last log file entries are parsed. Afterwards the log file parsing for the current log file is initiated.

The class LogFileParser is concerned with the log file parsing. The member variables are described in Table 4.4.

TABLE 4.4: Member Variables Class LogFileParser

Variable	**Type**	**Explanation**
_client_log_file	STRING	name of XML file

Continued on next page

Table 4.4 Member Variables - *continued from previous page*

Variable	Type	Explanation
_client_log_file_fd	INTEGER	file descriptor of XML file
_first_line	STRING	first lines of SRB log file
_last_byte_number	INTEGER	save last byte number which was parsed
_line_number	INTEGER	last line number which was parsed
_verbose	INTEGER	defines printing of debug messages
_ignore_error	INTEGER	error which are to be ignored
_keywords	STRING	array with keywords
_logfilepath	STRING	location (path) of the SRB log file

The constructor __init__ initialises the member variables. The function set_first_lines save the first fifteen lines of the log file. _fetch_first_lines only reads the first fifteen lines of the log file. With the function test_first_lines it is determined if a log file rotation took place. get_first_lines returns the content of the member variable _first_line. The function reset reset the member variables _line_number and _last_byte_number after a log file. To be able to delete the last tag within the XML file the size in bytes is determined by the function find_size_last_message_tag. The functions start_entry, end_entry, and write_entry are used to write the XML file. The corresponding member variable is updated with update_keywords and update_ignore_errors, respectively. The recursive function _test_keywords determines if a log file line is taken or not taken. From the log file line the time is extracted with _extract_time and the error number is determined with _extract_error_number. The most important function is analyse_logfile. The program flow chart of the most important loop is displayed in Figure 4.4.

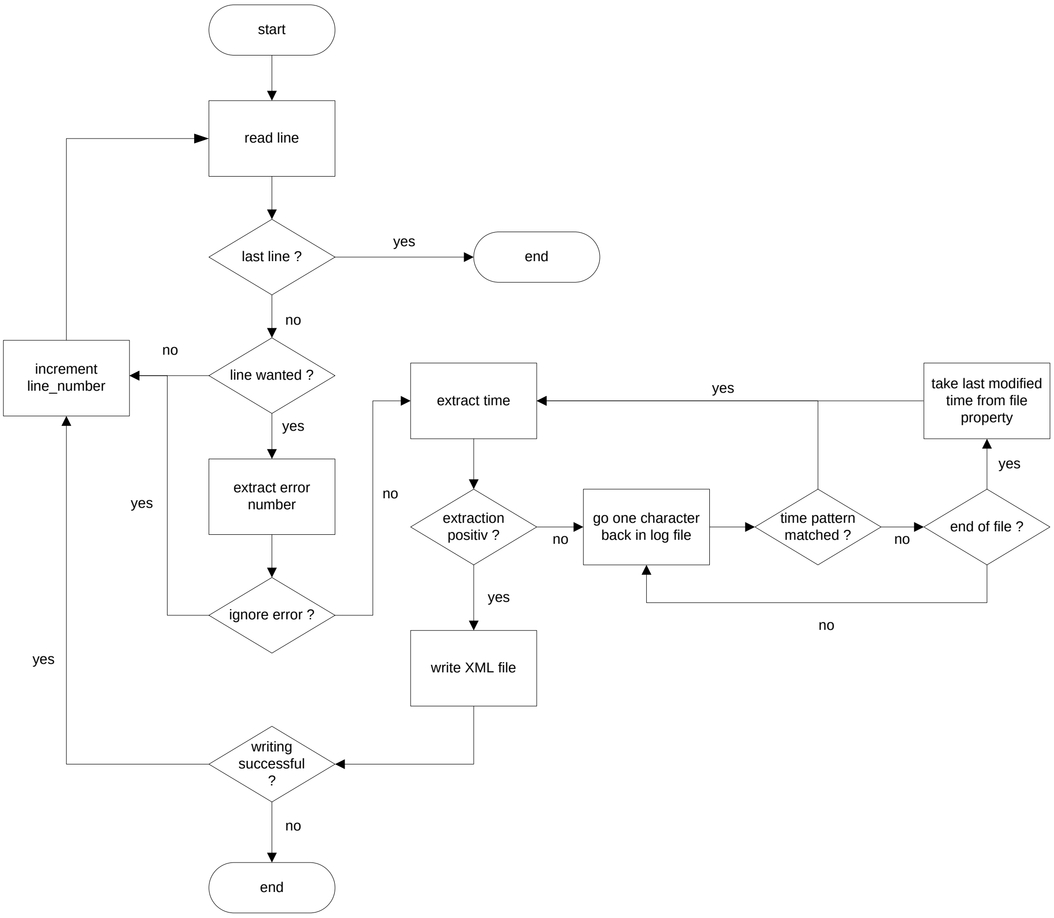


FIGURE 4.4: Flow Chart `analyse_logfile`

The parser read the log file line by line. If the end of the file is reached the parsing process is terminated as well as if problem occurs writing the XML file, *e.g.* if no hard disk space is available anymore. From the log file line the time is extracted. If this is not possible the log file properties are taken into account.

All the necessary RPC functions are centralised in the class `RPC`. Table 4.5 presents the member variables of the `RPC` class.

TABLE 4.5: Member Variables Class RPC

Variable	Type	Explanation
_verbose	INTEGER	defines printing of debug messages
_client	BOOL	indicates RPC status (enabled/ disabled)
_share	MUTEX	object of class Mutex
_config_file	STRING	name of configuration file
_interval	INTEGER	parsing interval time
_keyword_path	STRING	location (path) of keyword file
_keyword_name	STRING	name of keyword file
_xml_file_path	STRING	location (path) of XML file

The constructor __init__ initialises the member variables. The function rpc_stop_server executes the bash (bourne again shell) script to shut down the server. A detailed description about this script can be found in chapter 5.4. rpc_disable_rpc_calls and rpc_enable_rpc_calls are used to modify the member variable _client whereas rpc_status only returns to current value of the variable. The current parsing interval time can be discovered with the function rpc_interval_status. If the server is parsing the log file, the client has to wait until the server is finished. With the function rpc_check_availability it is possible to check if the server has finished the parsing process. rpc_get_file_list in conjunction with _parse_directory returns a list of all files, which are available for the client to fetch. Finally, the function rpc_get_my_xml_file delivers the XML file. It is possible the modify remotely the configuration and keyword file. The functions rpc_update_configuration and rpc_update_keyword_file enable this. Both functions work after the same structure. Different modes such as add, delete, or information are possible. According to the mode the corresponding file is modified or the required information gets extracted. During the file modification part of the file gets deleted and after exchanging or deleting the required value, rewritten.

The class SimpleSSLXMLRPCServer is derived from SimpleXMLRPCServer which is part of Python's standard library and SSL.Server which is provided by the M2Crypto package. This class implements the basic server. Table 4.6 shows the member variables.

TABLE 4.6: Member Variables Class `SimpleSSLXMLRPCServer`

Variable	Type	Explanation
`_verbose`	INTEGER	defines printing of debug messages
`_funcs`	STRING	dictionary for the RPC functions
`_logRequests`	INTEGER	defines if requests should be logged
`_instance`	*undetermined*	the class allows to install instances, this is not used for this project and therefore, it is set to "`None`"

The constructor `__init__` initialises the member variables and initialises the secure SSL server. With `handle_request` the original function of `BaseServer` is overwritten. For a better understanding parts of the derivation path can be illustrated as following:

$$\begin{tikzcd}
\texttt{BaseServer} \arrow[d, Rightarrow] \\
\texttt{SocketServer.TCPServer} \arrow[d, Rightarrow] \\
\texttt{SimpleXMLRPCServer} \arrow[d, Rightarrow] \\
\texttt{SimpleSSLXMLRPCServer}
\end{tikzcd}$$

With the new function `handle_request` every incoming request is passed on to an object of `MyClientThread`. This enables multithreading. A more detailed description can be found in chapter 5.2.

`MyClientThread` is derived from `threading.Thread`. Table 4.7 displays the member variables.

TABLE 4.7: Member Variables Class MyClientThread

Variable	Type	Explanation
_request	SOCKET	accepted request
_client	STRING	IP address from connecting client
_serv	SIMPLESSL-XMLRPC-SERVER	object of the current running SimpleSSLXMLRPCServer

The constructor __init__ initialises the member variables. The redefinition of the run function executes the in class BaseServer defined functions process_request and close_request.

The class My_SSL_Server implements the final server. The member variable hold the server certificate file name (STRING), certificate authority file name (STRING) and verbose mode (INTEGER). The constructor __init__ initialises the member variables. Within start_server the server get initialised and finally started. The function init_context provides the necessary SSL context.

The Mutex class handles the thread synchronisation. Table 4.8 illustrates the member variables.

TABLE 4.8: Member Variables Class Mutex

Variable	Type	Explanation
_parsing	INTEGER	indicates if server is busy
_client	INTEGER	indicates if client is busy
_parsing_thread_id	INTEGER	identification number of parsing thread
_locked	THREADING.L(	object of threading.Lock

The constructor __init__ initialises the member variables. The function set_variable_parsing ensures the work of the parsing thread and the function `set_variable_client` handles the synchronisation of the client threads. A more detailed description about the mutex mechanism can be found in chapter 5.6.

4.3 Client

The basic design of the client as illustrated in Figure 4.2 has a preprocessor for verifying the input. Afterwards the XML processor gets started and works independently. The processor is controlled by the manager thread. The actual connecting to the server is done by the parsing thread, which also takes care of processing the XML file (storing the preprocessed information in a database) and email notification.

The client is working with a configuration file in the same way the server does. Following issues are configurable

- the location and name of the database
- the location of
  - error description file
  - server certificate file
  - certificate authority file
- the XML fetching interval time
- the server IP in connection with the port
- SMTP mail server issues (server address, user name, sender's name)
- mail recipient issues (email address, location of keyword list file, error to be ignored)

The client fetches the prepared XML file from the server. Is the file successfully transferred to the client, the XML file on the server is deleted to avoid unnecessary memory usage. Several servers can be checked at the same time. This is realised with threads. Thread synchronisation is ensured with a mutex class object, which is passed on to each thread.

A thread connects with a dedicated server and checks if an XML file is available. If this is the case, the file is transferred to the client and saved temporarily on local disk. If the database is accessible, the XML file is parsed and XML entry by XML entry is stored in the database. The standard error numbers are provided by the error description file. The XML entry can provide such an error number. If no error number is provided by the XML file, the error message gets assigned the error number 999999. Any new error number is automatically inserted in the database. Double entries are avoided by checking the database beforehand if the entry already exists. Double entries can occur if the final XML processing or XML fetching process is interrupted and the client deals a second time with the same file.

At the same time a temporary mail content file is written. In the configuration file recipients can be defined, who receive a mail notification. The contents of the notification can be modified with additional keywords as well as with additional error numbers. The keywords, written in a keyword file, contain all those keywords, where the recipient is not interested in notification. Furthermore, it is possible to define certain error numbers, which are to be ignored and no notification is sent. All the content of the other XML entries are added to the mail content file. After the XML file was successfully parsed, the temporary XML file gets deleted. This is followed by creating a mail using the mail content file and sending the mail via SMTP. For that the module *smtplib* from Python standard library is used. One mail is send for each server monitored and each XML file fetched. The temporary mail content file is deleted afterwards.

For the authentication at the SMTP mail server a password is needed. This password can not be saved in any configuration file due to security issues. Also, to save an encrypted password locally is not an option, since the Python scripts (source code) are stored as plain text and easily accessible. Therefore, the decryption algorithm can be seen. The only possibility to gain a certain degree of security is to enter the password during the start process of the client. The password is then stored in the virtual memory for the time the application is running. For that the console echo is turned off, the password can be entered without appearing as console output. Afterwards the console echo is turned on again.

Each client has its own database, which gets initialised during the starting process.

The client can be run as a daemon. The application is daemonised as analysed in chapter 3. The configuration file is passed on as a parameter. Table 4.9 defines the parameter for the client. The work of the client can be monitored as console output. If the client is running as a daemon, the output is redirected into a log file. The log file is cleared each time the client is started.

TABLE 4.9: Client Parameters

Parameter	Explanation
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]
-p or –smtp_password	activates mail notification sending
-d or –daemon	daemonize the client

4.3.1 Client Class Diagram Design

The class diagram of the client application is shown in Figure 4.5.

The manager class `MyClient` is verifying the user input and initialising the application. Table 4.10 displays the member variables.

TABLE 4.10: Member Variables Class `MyClient`

Variable	Type	Explanation
`_verbose`	INTEGER	defines printing of debug messages
`_client_certificate`	STRING	name of the client certificate
`_client_certificate_path`	STRING	location (path) of the client certificate
`_client_ca`	STRING	name of certificate authority file
`_client_ca_path`	STRING	location (path) of the certificate authority file
`_error_description_name`	STRING	name of the error description file

Continued on next page

Table 4.10 Member Variables - *continued from previous page*

Variable	Type	Explanation
`_error_description_path`	STRING	location (path) of the error description file
`_workingpath`	STRING	name of working directory
`_database_name`	STRING	name of the database
`_database_path`	STRING	location (path) of the database
`_interval`	INTEGER	parsing interval period in minutes
`_project`	STRING	name of SRB project
`_serverlist`	STRING	array of server which are monitored
`_share`	MUTEX	object of class `Mutex`
`_smtp_server`	STRING	name of SMTP server
`_smtp_pass`	STRING	SMTP password
`_smtp_from`	STRING	email sender identification
`_smtp_user`	STRING	SMTP user name
`_mail_address`	STRING	notification email addresses
`_mail_ignore_error`	STRING	array of keywords
`_db`	MYDATABASE	object of class `MyDatabase`
`_workerthread`	WORKER-THREAD	object of class `WorkerThread`

The constructor `__init__` initialises the member variables. This function `initialise_database` is initialising the database. `get_serverlist` returns the content of the member variable `_serverlist`. With `fetch_error_messages` the workerthread is initialised and started. `_get_keywords` extracts keywords from a given file. The recursive function `_remove_item` deletes an item from a given list and is mainly used to delete comments which might be in a keyword file.

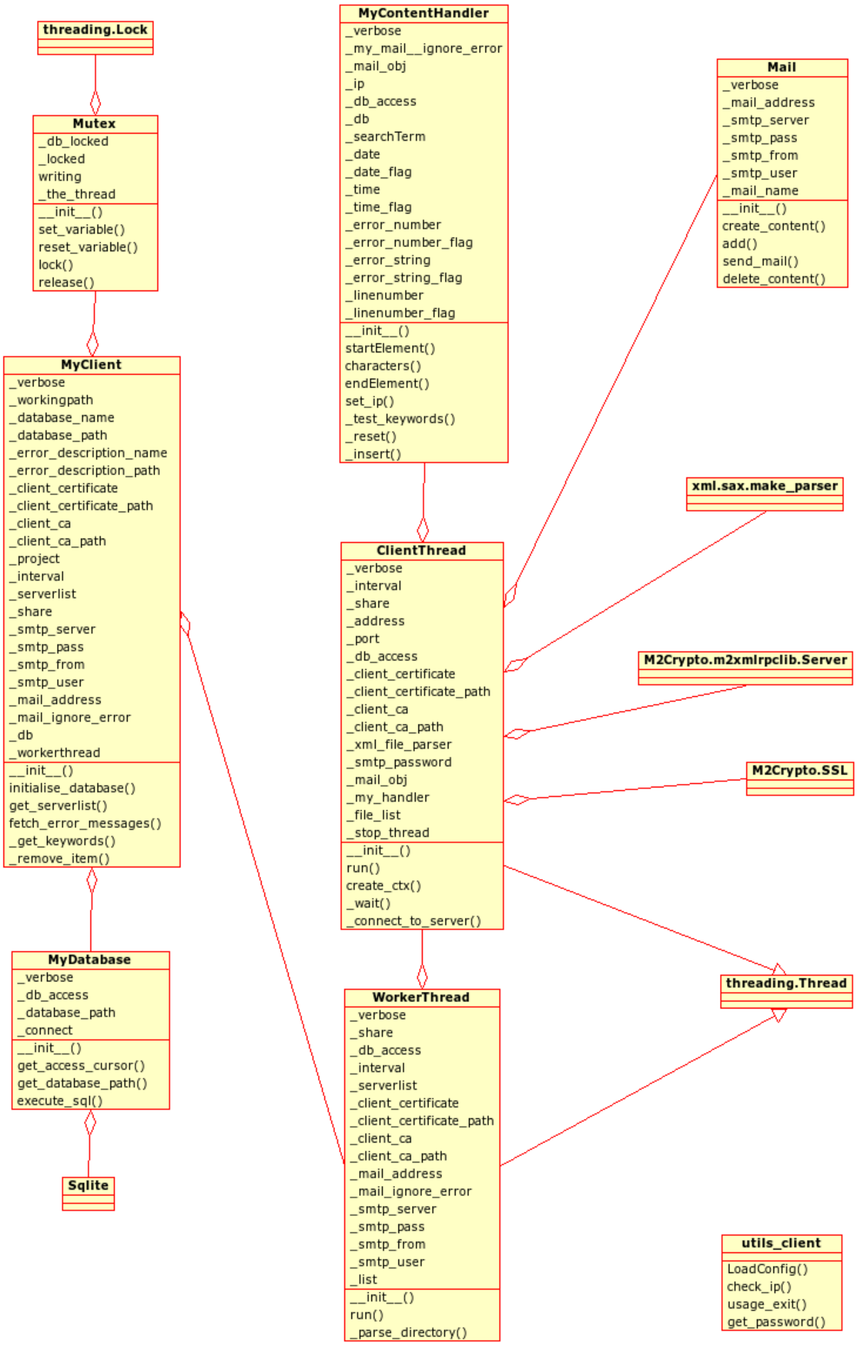


FIGURE 4.5: Client Class Diagram

The class `WorkerThread` is responsible for starting the threads which are connecting to the server and is derived from threading.Thread. The member variables are presented in Table 4.11.

TABLE 4.11: Member Variables Class WorkerThread

Variable	Type	Explanation
_verbose	INTEGER	defines printing of debug messages
_share	MUTEX	object of class Mutex
_interval	INTEGER	parsing interval period in minutes
_client_ca	STRING	name of certificate authority file
_client_ca_path	STRING	location (path) of the certificate authority file
_client_certificate	STRING	name of the client certificate
_client_certificate_path	STRING	location (path) of the client certificate
_serverlist	STRING	array of server which are monitored
_smtp_server	STRING	name of SMTP server
_smtp_pass	STRING	SMTP password
_smtp_from	STRING	email sender identification
_smtp_user	STRING	SMTP user name
_mail_ignore_error	STRING	array of keywords
_mail_address	STRING	notification email addresses
_db_access	MYDATABASE	object of class MyDatabase
_list	STRING	array to hold a file names

The constructor __init__ initialises the member variables. The function _parse_directory is used with the function os.path.walk. This function "walks" through a given directory and considers all srbLOG*.gz files. The name and last modified time are saved in a two dimensional array. Finally, the function run initiates the periodically fetching and processing of the XML files.

ClientThread derived from threading.Thread as well is handling the actual XML fetching and processing in connection with Mail. Table 4.12 displays the member variables.

TABLE 4.12: Member Variables Class ClientThread

Variable	**Type**	**Explanation**
_verbose	INTEGER	defines printing of debug messages
_share	MUTEX	object of class Mutex
_interval	INTEGER	parsing interval period in minutes
_client_ca	STRING	name of certificate authority file
_client_ca_path	STRING	location (path) of the certificate authority file
_client_certificate	STRING	name of the client certificate
_client_certificate_path	STRING	location (path) of the client certificate
_address	STRING	IP address of server which are monitored
_port	INTEGER	port number of server
_smtp_password	STRING	SMTP password
_mail_obj	MAIL	object of class Mail
_smtp_user	STRING	SMTP user name
_db_access	MYDATABASE	object of class MyDatabase
_my_handler	MYCONTENT-HANDLER	object of class MyContentHandler
_stop_thread	BOOL	indicates manually terminating of thread
_file_list	STRING	array to hold a file names

Continued on next page

Table 4.12 Member Variables - *continued from previous page*

Variable	Type	Explanation
_xml_file_parser	XML.SAX. MAKE_PARSER	object of class xml.sax.make_parser

The constructor __init__ initialises the member variables. The XML parser require a content handler which is provided by MyContentHandler. The necessary SSL context to connect with the server is supplied by create_ctx. _connect_to_server establishes the secure connection to the server. While the server is parsing the SRB log file, the function _wait checks for a defined time if the XML file can be fetched. Within run the whole XML file fetching and processing procedure is executed.

The fetching consists of three parts. Figure 4.6 illustrates the top level flow chart diagram as an overview of part I to III. After the connection is successfully established (part I) the client determines which files need to be fetched (part II). If XML files on the server side available the actual fetching takes place (part III).

All these parts contain routines for following scenarios

- RPC is disabled
- server is busy
- server is not reachable

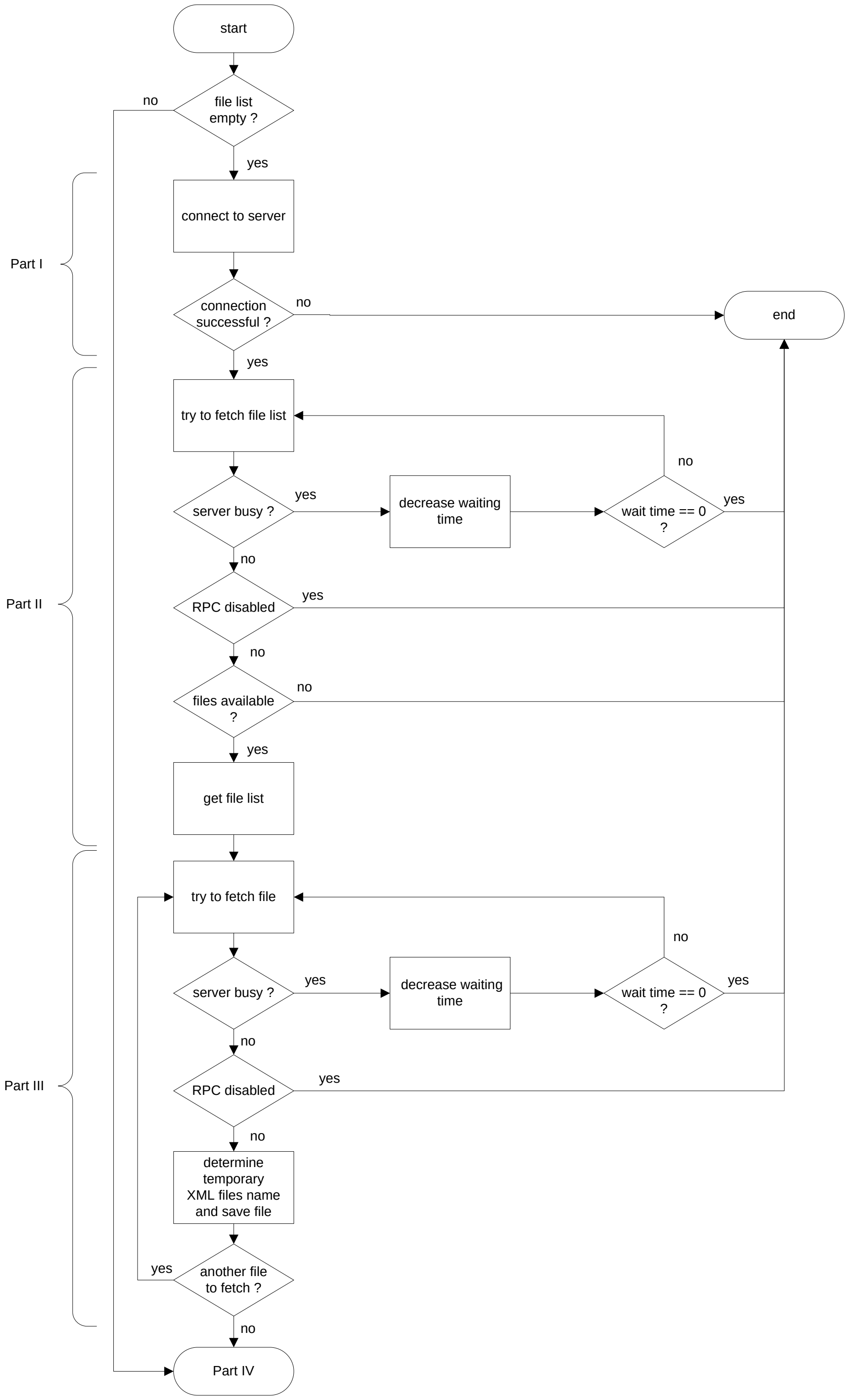


FIGURE 4.6: Flow Chart `ClientThread - run()` Part I - III

Now the XML processing is executed (part IV). The temporary saved files contains the IP address from producing server. Figure 4.7 shows a top level flow chart diagram of part IV.

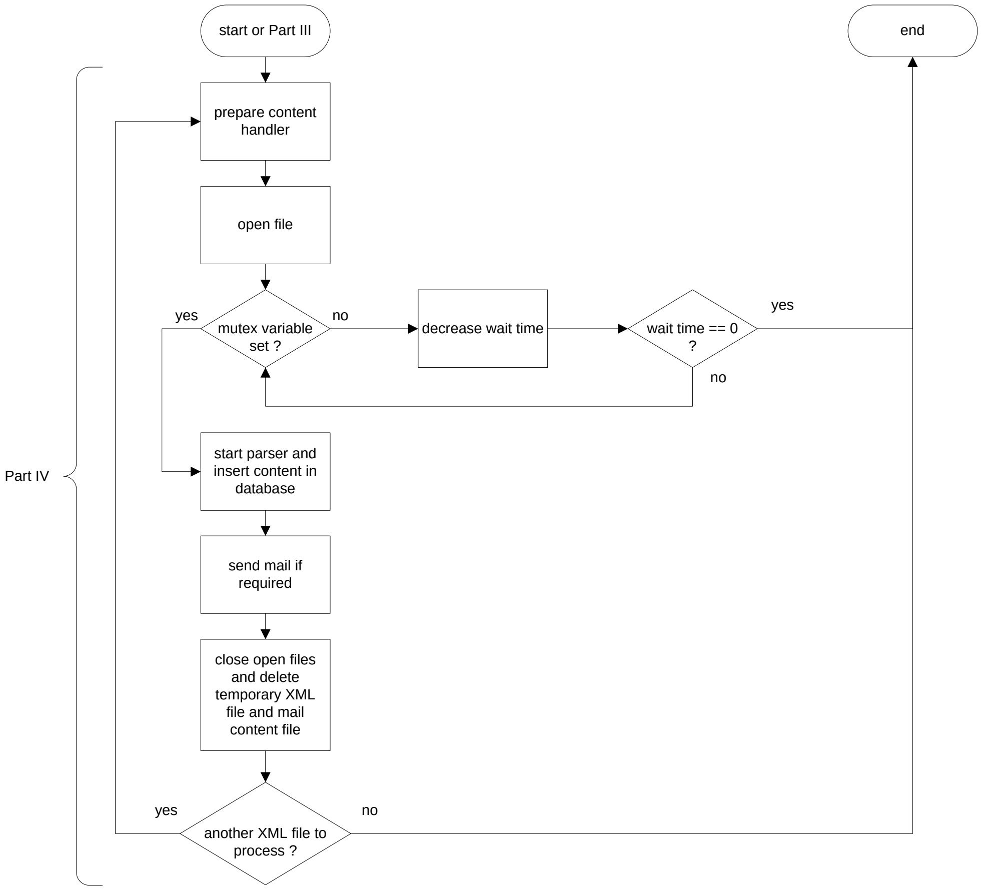


FIGURE 4.7: Flow Chart `ClientThread - run()` Part IV

First the content handler is prepared. Then the parser is started and if required the mail is sent. Completed is the procedure with the deleting of all temporary files. The function can also be used to process locally saved XML files only, if a list of files is passed on as a parameter already.

The XML parser module has to know how to manage the content of the XML file. This is done with `MyContentHandler` which is derived from `xml.sax.handler.ContentHandler`. Table 4.13 presents the member variables.

TABLE 4.13: Member Variables Class MyContentHandler

Variable	Type	Explanation
_verbose	INTEGER	defines printing of debug messages
_my_mail- _ignore_error	STRING	array of keywords
_mail_obj	MAIL	object of class Mail
_ip	STRING	server IP address
_db_access	SQLITE	database access cursor
_db	MYDATABASE	object of class MyDatabase
_searchTerm	STRING	tag which needs to be identified
_date	STRING	XML content for date
_date_flag	INTEGER	indicates if date content is found
_time	STRING	XML content for time
_time_flag	INTEGER	indicates if time content is found
_error_number	STRING	XML content for error number
_error_number_flag	INTEGER	indicates if error number content is found
_error_string	STRING	XML content for error string
_error_sting_flag	INTEGER	indicates if error string content is found
_linenumber	STRING	XML content for line number
_linenumber_flag	INTEGER	indicates if line number content is found

The constructor __init__ initialises the member variables. The function startElement defines the XML tag which is handled. If a tag is matched, the function characters assigns the content to the appropriate member variable. If all flags are set, endElement initialises the database update and mail content writing. The actual writing into the database is executed with _insert where also all necessary verifications takes place, *e.g.* double entry check. For the mail content creating the recursive function _test_keywords determines the actual mail content. The function _reset is used to reset member variables. The variable _ip can be modified with set_ip.

MyDatabase handles database issues like initialising and updating as well as providing a database access cursor. The member variable are presented in Table 4.14.

TABLE 4.14: Member Variables Class MyDatabase

Variable	Type	Explanation
_verbose	INTEGER	defines printing of debug messages
_db_access	SQLITE	database access cursor
_database_path	STRING	location (path) of database file
_connect	SQLITE	object of class sqlite

The constructor __init__ initialises the member variables and creates or updates the database. Any database corruption is also detected here. Figure 4.8 gives an overview about the constructor structure.

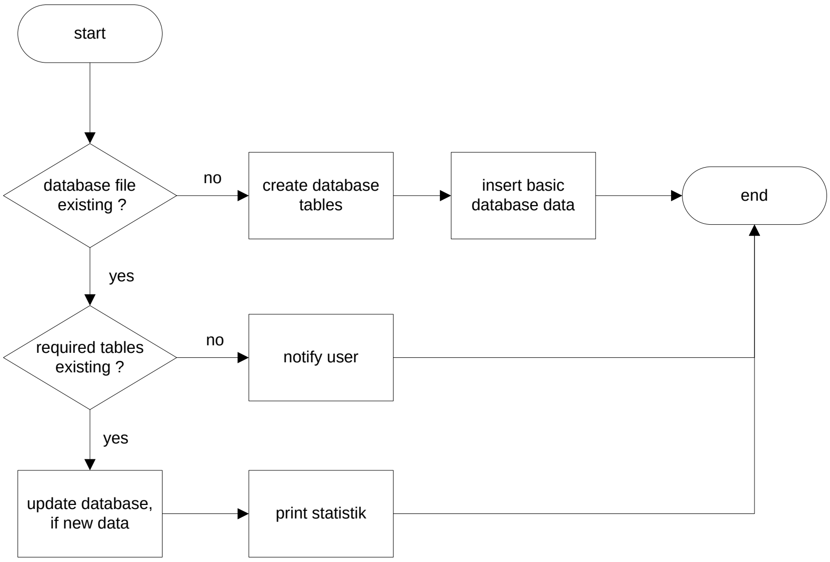


FIGURE 4.8: Flow Chart Constructor MyDatabase

The database is created and the basic data such as error from the error number file, if no database file exists. If a database file is detected, each table is verified. Every irregularity is reported. Finally a statistic about the current database state is printed. Any new data is inserted automatically, *e.g.* new server IP addresses. The functions `get_access_cursor` and `get_database_path` return the content of the corresponding member variable. Finally, the `execute_sql` executes a SQL command.

Instruments to send a mail are provided by the class `Mail`. Table 4.15 diplays the member variable.

TABLE 4.15: Member Variables Class `Mail`

Variable	**Type**	**Explanation**
`_verbose`	INTEGER	defines printing of debug messages
`_mail_access`	STRING	receiver address
`_smtp_server`	STRING	SMTP server address
`_smtp_pass`	STRING	SMTP password
`_smtp_from`	STRING	email sender identification
`_smtp_user`	STRING	SMTP user name
`_mail_name`	STRING	name of temporary mail content file

The constructor `__init__` initialises the member variables. With `create_content` the temporary mail content file is created. The function `add` is used to add information to the content file. The content file is deleted with `delete_content`. The mail is sent with the function `semd_mail` using the `smtplib.SMTP` from Python's standard library.

`Mutex` is used for thread synchronisation. Table 4.16 present the member variables.

TABLE 4.16: Member Variables Class Mutex

Variable	Type	Explanation
_writing	INTEGER	indicates a thread is writing the database
_the_thread	INTEGER	identification number of writing thread
_db_locked	THREADING. LOCK	object of threading.Lock for database synchronisation
_locked	THREADING. LOCK	object of threading.Lock for any other occurring critical section

The constructor __init__ initialises the member variables. The function set_variable set the member variable _writing and the function rest_variable resets this variable. With lock and release the lock _locked can be operated.

4.4 Database Design

The values of the XML file as defined in section 4.2 have to be stored in a database. Furthermore, all the existing error codes as well as the server which are monitored have to be saved.

The design of a database can be presented as an Entity Relationship Model (ERM). An ERM is a conceptual data model to view the reality as entities and relationships between entities. An entity is the data object, which contains the data to be stored. It consists of attributes and is analogue to the table in the relational database. Attributes describe the entity. Each attribute has a domain. The domain defines all possible values an attribute can have. Relationships between entities can be classified in many ways. Cardinality is one possibility and following relations can be committed:

- **1:1**
  one instance of entity A is associated with only one instance of entity B

- **1:n**
  one instance of entity A is associated with zero, one, or many instances of entity B

- **n:m**
  one instance of entity A is associated with zero, one, or many instances of entity B and one instance of entity B is associated with zero, one, or many instances of entity A

The relations can be presented within the model using symbols as illustrated in Figure 4.9:

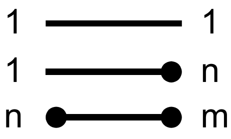


FIGURE 4.9: Cardinality within ERM

Figure 4.10 shows an extended Entity Relationship Model for the required database. The extended ERM defines precisely the range of possible values (min, max).

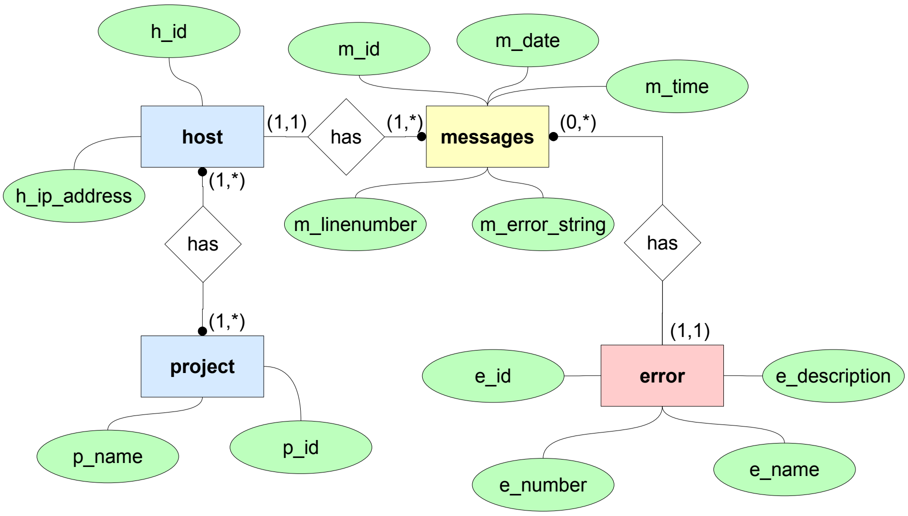


FIGURE 4.10: Entity Relationship Model

Based on the ERM, Figure 4.11 illustrates the design of the database.

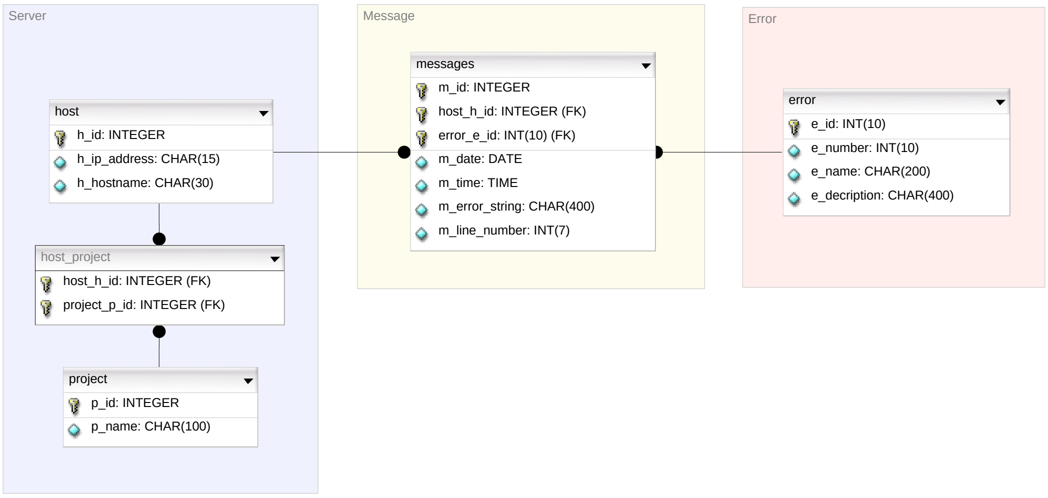


FIGURE 4.11: Database Design

The table `messages` is storing the XML values and is shown in Table 4.17.

TABLE 4.17: Database Table `messages`

Column Name	Data Type	Description
m_id	**INTEGER**	**unique primary key** (autoincrement)
host_h_id	INTEGER	foreign key
error_e_id	INTEGER(10)	foreign key
m_date	DATE	date of the error occurrence
m_time	TIME	time of the error occurrence
m_error_string	CHAR(400)	error message (log file line)
m_line_number	INTEGER(7)	line number within the SRB log file

The values `host_h_id` and `error_e_id` form the connections to the tables `host` and `error`. The attribute `m_id` serves as unique primary key. A message has only one error number.

The table error with its attributes is listed in Table 4.18.

TABLE 4.18: Database Table error

Column Name	Data Type	Description
e_id	**INTEGER**	**unique primary key** (autoincrement)
e_number	INTEGER(10)	error number
e_name	CHAR(200)	error name
e_description	CHAR(400)	error description

An error number can be assigned to many messages. The table host (Table 4.19) keeps track of the monitored server. A server can be assigned to many messages.

TABLE 4.19: Database Table host

Column Name	Data Type	Description
h_id	**INTEGER**	**unique primary key** (autoincrement)
h_ip_address	INTEGER(10)	IP address of the server

A certain message can only be connected to one particular server. It is possible to create a SRB project, which can be spread over several servers, although for this project it is defined that a server has only one project. The connection table (Table 4.21) is needed to connect the servers with projects (Table 4.20).

TABLE 4.20: Database Table `project`

Column Name	Data Type	Description
`p_id`	**INTEGER**	**unique primary key** (autoincrement)
`p_name`	INTEGER(10)	SRB project name

TABLE 4.21: Database Table `host_project`

Column Name	Data Type	Description
`p_id`	INTEGER	foreign key
`hp_p_id`	INTEGER	foreign key

4.5 Virtualiser

Since the client as the application with database access is able to run as a daemon it should not be used to display the database content. Therefore another tool is developed - the “Virtualiser”.

The Virtualiser is querying the database only and is located at the same system like the database. Table 4.22 contains a summary of all needed queries.

TABLE 4.22: Database Queries

Query	Expected Answer
find all projects	return a list of projects
find all hosts	return a list of hosts and the projects they belong to
find all errors between date X and date Y	return a list of errors, dates, hosts, projects

Continued on next page

Table 4.22Database Queries - *continued from previous page*

Query	Expected Answer
find all errors between date X and date Y for project Z	return a list of errors, dates, hosts
find all errors between date X and date Y on host Z	return a list of errors, dates, projects
find all errors of type X	return a list of hosts, projects, errors, dates, errors
find all errors of type X between date X and date Y	return a list of hosts, projects, errors, dates

The console output is coloured to support the tool usage. The table 4.23 defines the parameter for the display tool.

TABLE 4.23: Virtualiser Parameters

Parameter	Explanation
	general parameters
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]
-g or –graph	show output additionally as a diagram
–nocolor	no coloured console output
–file <string>	dump output into a file (file name has to be given)
	database commands
–sql_host	show all hosts
–sql_project	show all projects
–sql_error	show errors (additional parameters possible)
–sql_error_freq	show only frequency of errors (additional parameters possible)
	additional parameters

Continued on next page

Table 4.23 Virtualiser Parameters - *continued from previous page*

Parameter	Explanation
–start_date <date>	start date (*e.g.* 23.12.2005)
–end_date <date>	end date (*e.g.* 23.01.2006)
–start_time <time>	start time (*e.g.* 23:12:19)
–end_time <time>	end time (*e.g.* 23:12:59)
–ip <ip>	host IP (*e.g.* 127.0.0.1)
–project <string>	specify a certain project
–error <int,int...>	specify a certain error (comma separated list)

To summarise, the user is able, through a combination of parameters, to gain the needed information from the database. By default the query results are printed in the console. The console output can be also directed into a file. If desired, the results are displayable with a graph as a function of frequency. For missing dates, *e.g.* for a particular error is no data available for a certain date which lies in between the start and end data, the error occurance value zero will be inserted. This is necessary to gain a complete view. Without the insertion the graph will be misleading. To display the graph a pop-up window is generated. The window contains following basic components

- graph (bar chart or line chart) including description of the axes
- plot button, to zoom and generated a new window
- save button, to save graph as postscript file
- quit button, to close window

To establish usability a status bar which displays a short description about the currently used window element is included. A dialog to lead the user through the saving process has to be built. To avoid accidentally closing of the windows a message box is needing to inform the user about to event which is going to happen.

The class `Display` is evaluating the given parameters and querying the database. If a graph is needed the querying results are passed on to an object of `Picture`. For each graph an individual object is created. The window creation is done with modules of the Tkinter interface. The class Colour is used to colour the console output and is described in detail in chapter 5.8.

4.5.1 Virtualiser Class Diagram Design

Figure 4.12 presents the class diagram for the Virtualiser application.

The managing class is called Display. The member variables are shown in Table 4.24.

TABLE 4.24: Member Variables Class Display

Variable	Type	Explanation
_database_name	STRING	name of the database file
_database_path	STRING	location (path) of the database file

The constructor __init__ verifies the user input and initialises the member variables. The functions sql_host and sql_project provide certain static SQL commands. sql_error offers a very flexible SQL command. With this function most of the possible database queries can be executed. The final SQL command depends on the given parameters.

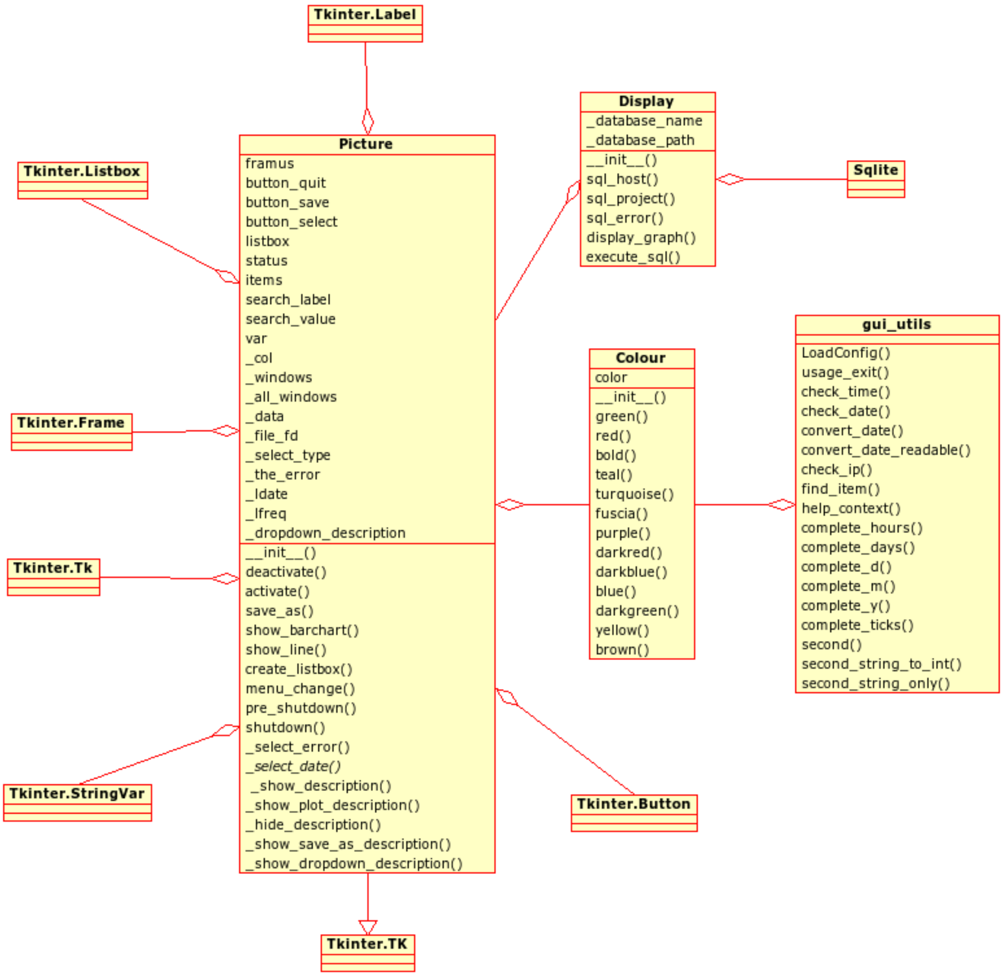


FIGURE 4.12: Virtualiser Class Diagram

Function `execute_sql` is used to execute the SQL commands. As a special feature of this function is the possibility to pass on a time. If the database is not available the function will try to execute the SQL command for the defined time.

The class `Picture` is handling the graphical user interface. The class structure is very flexible so that bar charts and line diagrams can be created. Table 4.25 contains the member variables.

TABLE 4.25: Member Variables Class `Picture`

Variable	Type	Explanation
`framus`	TKINTER.FRAME	object of class `Tkinter.Frame` which provides the basic frame
`button_quit`	TKINTER.BUTTON	object of class `Tkinter.Button` which realises the “quit” button
`button_save`	TKINTER.BUTTON	object of class `Tkinter.Button` which realises the “save as” button
`button_select`	TKINTER.BUTTON	object of class `Tkinter.Button` which realises the “plot” button
`listbox`	TKINTER.LISTBOX	`Tkinter.Listbox`
`status`	TKINTER.LABEL	`Tkinter.Label` to realise the status bar
`items`	STRING	items which are displayed in the graph
`search_label`	STRING	labels of x-axis
`search_value`	STRING	values of x-axis
`var`	TKINTER.STRINGVAR	object of class `Tkinter.StringVar` to modify listbox
`_col`	INTEGER	indicates of colored output is required
`_windows`	PITCURE	array of children objects of `Picture`
`_all_windows`	PITCURE	array of all created objects of `Picture`
`_data`	STRING	array of SQL query results
`_file_fd`	INTEGER	file descriptor of file to save console output
`_select_type`	STRING	graph type

Continued on next page

Table 4.25 Member Variables - *continued from previous page*

Variable	Type	Explanation
_the_error	INTEGER	chosen error which is examined closer
_ldate	STRING	sorted date listbox value
_lfreq	STRING	sorted error listbox value
_dropdown-_description	STRING	description for dropdown menu which appears in status bar

The constructor __init__ initialises the member variables and creates the basic window with all the buttons. The functions deactivate and activate enable and disable buttons if a message box or additional dialog is opened. With save_as a dialog, provided by tkFileDialog.asksaveasfilename, is executed. This dialog leads the user through the saving process. The listbox is created and initialised with create_listbox. The order within the listbox can be influenced with _menu_change. In conjunction with the listbox the functions _select_error and _select_date are used to extract and process the chosen item from the listbox. pre_shutdown and shutdown are used to close the windows, whereas pre_shutdown provokes a message box to inform the user about the upcoming action. The functions show_description, show_plot_description, show_save_as_description, show_dropdown_description, and _hide_description are used to modify the status bar. The actual graph are produced with show_barchart and show_line. Both function use the module Graph.py which contains classes and related methods necessary to create graph widgits. The code of Graph.py is taken from an example presented in the book "Python and Tkinter Programming" by John E. Grayson [21] and later modified by Dr. Adil Hasan and the author.

4.6 Remote Controller

To give the user the possibility to adjust the server configuration remotely the tool "Remote Controller" is developed. The Remote Controller is a console application as well and the parameter definitions in Table 4.26 show among other parameters those elements which can be influenced on the server side.

TABLE 4.26: Remote Controller Parameters

Parameter	Explanation
	general parameters
-h or –help	print help
-c or –config	defines configuration file
-g or –graph	show output additionally as a diagram
–nocolor	no coloured console output
	server commands
–rpc_status	show actual setting of RPC (disabled/enabled)
–disable_rpc	disable RPC
–enable_rpc	enable RPC
–shutdown	shutdown server
–change_interval <int>	change parsing interval of server
–keyword_status	show actual setting of keywords
–add_keyword <string>	add keyword to keyword list
–delete_keyword <string>	delete keyword in keyword list
–ignore_error_status	show actual setting of "ignore_error"
–add_ignore_error <int>	add error, which the parser should ignore
–delete_ignore_error <int>	delete error, which the parser is ignoring
	additional parameters
–ip <ip>	host IP (*e.g.* 127.0.0.1)
–port <int>	port, where the server is listening

The Remote Controller is a very small tool and has only one important class - `Admin`. Figure 4.13 shows the class diagram for the Remote Controller. The class `Colour`, already used for the Virtualiser, is needed to colour the console output. This is a special feature for usability and is descriped more closely in chapter 5.8.

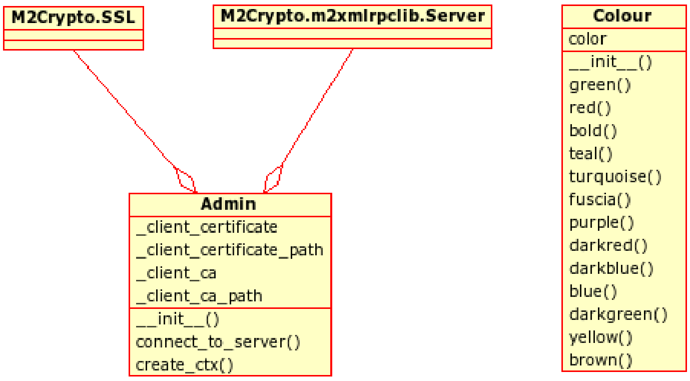


FIGURE 4.13: Remote Controller Class Diagram

Table 4.27 presents the member variables for the class `Admin`.

TABLE 4.27: Member Variables Class `Admin`

Variable	Type	Explanation
`_client_certificate`	STRING	name of client certificate file
`_client_certificate_path`	STRING	location (path) of client certificate file
`_client_ca`	STRING	name of certificate authority file
`_client_ca_path`	STRING	location (path) of certificate authority file

The constructor `__init__` verifies the user input and initialises the member variables. The function `connect_to_server` establishes the connection with the server. The necessary SSL context is provided by `create_ctx`.

4.7 GZ Parser

The SRB log file analysis showed that due to log file rotation older log files are saved compressed in another location. Since those files have to be parsed once only, a separate tool is developed - the “GZ Parser”.

Table 4.28 shows the possible parameter of GZ Parser.

TABLE 4.28: GZ Parser Parameters

Parameter	Explanation
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]

The GZ Parser uses the same module for evaluating the log file and creating an XML file as the server (class `LogFileParser`, described in chapter 4.2.1). Due to flexibility a configuration file is used already described section 4.2. Following issues can be configured in the configuration file:

- location and name of keyword file
- location of *.gz files
- location of director, where to store the XML files
- errors to be ignored

The user ensures by adjusting the configuration file, that the XML file is placed in the directory of the server, where the client can fetch those files. If this directory already contains an XML file, overwriting of this file is avoided by renaming the new XML file. The new XML file has the same name including a number. The number is increasing with each new XML file.

5 Implementation

In this chapter some aspects concerning the implementation of the previous described design are elaborated. Faced problems and corresponding solutions are highlighted.

During the software development the object oriented programming paradigms mostly were followed. It was tried to write independent modules to reuse the modules for similar purposes. The development can be referred as “agile” software development. “Agile” in this case means being flexible in all directions. This software development technique follows the principles (manifesto)

- Individuals and interactions over processes and tools [22]
- Working software over comprehensive documentation [22]
- Customer collaboration over contract negotiation [22]
- Responding to change over following a plan [22]

Considering these principles the customer satisfaction and having always a running software product have a high priority. Changes are accepted at any time. Behind the idea of agile software development many methods are hidden. This project used more or less the software management method scrum. Scrum concentrates more on the execution process. With regular meetings and setting certain scopes each time the development process was constantly supervised and changes could be applied immediately. Almost at each meeting a running software product could be presented.

For each application applies verifying user input is essential. Therefore all the parameters as well as the data read from the configuration files and keyword files are verified. This is usually done by the the constructors in the manager class of each application.

The complete source code is available in the appendix in chapter **??**. The class documentation was created with Doxygen [23] and can be found in chapter **??**.

5.1 General Aspects

The applications were developed on a SuSE 9.2 operating system. The software was designed and implemented for Python Version 2.2.3. To install Python and other software a C compiler is needed. The GCC is recommended. It is to be paid attention to the fact that Tkinter has to be enabled before compilation. Further, the following additional software packages were used to develop the applications and there are needed to run the software successfully

- M2Crypto Version 0.13 requires OpenSSL 0.9.7 and SWIG 1.3.2[123]
- sqlite Version 2.8.16
- pysqlite Version 1.0.1
- module `Graph.py`
- bash (bourne again shell)
- awk (Aho, Weinberger, Kernighan)
- egrep (extended global regular expression printer)

Most of the additional packages can also be installed with user rights. The script structure of all applications is very similar. Files called `*classes.py` contain most of the needed classes, file like `utils_*.py` contain additional small functions which are used from many classes. The start scripts usually contains the manager class and for the server and client the deamonise function.

5.2 SimpleSSLXMLRPCServer

Python's standard library comes with a module called `SimpleXMLRPCServer`. This module provides a basic server framework for XML-RPC servers [13]. The `SimpleXMLRPCServer` class is based on the `SocketServer.TCPServer` class, and the request handler is based on the `BaseHTTPServer.BaseHTTPRequestHandler` class [13].

The aim was to change the `SocketServer.TCPServer` into a secure TCP server. Hence a new class was created, derived from the `SimpleXMLRPCServer` and `SSL.Server`. `SSL.Server` is provided by the M2Crypto package. The new class `SimpleSSLXMLRPCServer` is shown in Listing 5.1.

LISTING 5.1: `SimpleSSLXMLRPCServer`

```python
class SimpleSSLXMLRPCServer(SSL.SSLServer, SimpleXMLRPCServer):
    '''
    overwrite the init function of the SimpleXMLRPCServer and replace it with the secure SSLServer
    '''
    def __init__(self, ssl_context, address, verbose, handler=SimpleXMLRPCRequestHandler):
        '''
        constructor
        '''
        SSL.SSLServer.__init__(self, address, handler, ssl_context)
        self.funcs = {}
        self.logRequests = 0
        self.instance = None
        self._verbose = verbose

    def handle_request(self, serv):
        '''
        handle one request and pass it on the a thread (enables multithreading)
        '''
        try:
            request, client_address = self.get_request()
            if self._verbose == 1:
                    print "%s -> request accepted from %s....." % (time.ctime(), client_address[0])

        except socket.error:
            return

        if self.verify_request(request, client_address):
            thd = MyClientThread(request, client_address, serv)
            thd.start()
```

Then the `init` function was redefined by overwriting the `SocketServer.TCPServer` with the `SSL.Server`.

To gain multithreading the request handler was overwritten as well. An incoming request gets accepted and forwarded to a thread. The thread dies after the work is finished.

5.3 The Parsing Approach

The success of the software written for this project depends on the correct evaluation of the given data, depends on the parsing module. Partly already written software was used, partly own ideas got implemented to meet the requirements.

The configuration files are parsed with the standard library module `ConfigParser`, as described in section 4.2. As the configuration file is modified by the user, it needs some simplicity. The chosen way, using this file structure, offers that.

5.3.1 Keywords

The user has a separate file to define keywords. These keywords are case sensitive. The first idea was to define all those keywords, which the user might be interesting in, a **positive approach**. But the user does not know what kind of messages or errors he might have to face. With the positive keyword approach he might lose important information. But what he can define more clearly is, in what messages or errors he is not interested. Hence the keyword approach was change to a **negative approach**.

During the initialising process the keywords are saved into an array. The entries of the array can be seen as `OR` combination. Each entry of this array can contain two items. The items are `AND` combinations. The character “`!`” serves as negator. For example following keyword file entry,

```
findServerExec, NOTICE:!error, NOTICE:!status
```

would mean that the user in **not** interested in lines which contain

$$[\,findServerExec\,] \vee [\,(NOTICE) \wedge (\neg error)\,] \vee [\,(NOTICE) \wedge (\neg status)\,]$$

The parser reads a line from the SRB log file. The line and the array is passed on to the recursive function `_test_keywords()`. The function goes through the array and if a keyword or a keyword combination is detected the function returns with the value `-1`, otherwise it returns `0`.

5.3.2 Line Processing

After the log file line was identified as intended, the error number gets extracted. Not all lines have an error number. In that case the character “-” is taken instead. Next step is extracting the date and time. Unfortunately, the log file entry does not contain a year. Hence the year is extracted from the log file properties itself. If no data and time is available the parser module goes back within the log file line by line until a date or time is found. If the top of the file is reached the needed data is extracted from the log file properties. Now that all the required information are gathered, the XML file is written. Finally the current line number is saved. Assuming this was the last line in the log file, in the next parsing period the parser can start exactly at that line and does not have to go through the parsed lines again. Then the parser module tries to read the next line from the log file.

5.4 How to Stop a Daemon

Client and server can be run as a daemon. Once the application is daemonised all terminals lost control over the daemon. But it is necessary to stop the application. The applications running in an UNIX operated environment. UNIX usually provides a certain list of tools to support the user. To stop a daemon a bash script was written. The bash (bourne again shell) is one of the oldest UNIX command shells. A shell serves the user as an interface to the operating system. Listing 5.2 shows the script for stopping the client daemon. The script to stop the server is analogue.

LISTING 5.2: Script `stop_client.sh`

```
#!/bin/sh
#
# Script to shutdown client daemon
#
# by Andrea Weise - December 2005
# University of Reading
# MSc in Network Centred Computing
#
echo "stopping client ...."

name=start_client.py

# Find all clients
client_pid=`ps -elf | egrep $name | egrep -v grep | awk '{ print $4 }'`

if [ "$client_pid" = "" ]
then
  echo No client is running !
else
```

```
20    /bin/kill -15 $client_pid
21    client_pid=`ps -elf | egrep $name | egrep -v grep | awk '{ print $4 }'`
22    if [ "$client_pid" = "" ]
23    then
24      echo client stopped
25    else
26      /bin/kill -9 $client_pid
27      echo client killed
28    fi
29  fi
```

Line 1 indicates, that the script should be executed by the bash. In line 11 the script name of the application which needs to be stopped is saved. Line 14 forms the heart of the script. With the command `ps` an instantaneous process table is created. The parameter `-e` invokes that every process is shown. The parameter `-l` activates the long output format. Parameter `-f` tries to gain as much information about the processes as possible. The sign `|` is the symbol for pipe and it connects two commands with each other. The output of the first command serves as input of the second command. Therefore the created process table is passed on to `egrep`. `egrep` stands for extended global regular expression printer and it searches for a given pattern. In this case all lines from the process table which contain the previous saved name are given. The output is again passed on to `egrep` with the parameter `-v grep`. This invokes that `egrep`'s own process, which would be part of the table is eliminated. Now the process ID is extracted by using `awk`. The name `awk` is assembled from the three creators of this programming language, Alfred V. Aho, Peter J. Weinberger and Brian W. Kernighan. It is used to evaluate text and is usually installed on UNIX systems. In Line 14 finally the 4th value of the extracted line, the process ID. This number is assigned to the variable `client_id`. If `client_id` is empty (line 16) no client was running and the script finishes. Otherwise the script is trying to terminate the process with the command `kill -15`, `SIGTERM` (line 20). After that the script checks again if the process is really gone. If the process is still running the `kill` command is executed again but this time with the parameter `-9`, `SIGKILL` (line 26).

5.5 XML

5.5.1 XML Creation

The server is generating an XML structured file. Not well formed files will influence the performance of the server as the server is an XML based server. Therefore, the creation of the XML has to ensure the final file is well formed. A DTD is not needed because the parser on the client side has a verifying content handler.

The DOM offers good possibilities to create such a file, because the whole structure is loaded into memory. Navigation through this structure is then easily possible and single nodes can be added fast and simple, since the DOM handles this. After implementing the file creation with DOM a performance test was run to evaluate the work of the DOM parser. A log file with size of 110 MB was taken and assumed each log file entry is an interesting error, where the information needs to be saved in the XML file. After 30 minutes the application had not finished parsing and the test was manually terminated. Since the log file was relatively large, the system was busy with managing this file and the new XML where the size was even larger, since additional information where added. The DOM kept reorganising and restructuring the constantly in size increasing XML file in memory. That slowed the whole system down. This result was unacceptable.

Therefore the XML file creating was rewritten, using SAX. SAX implementation of creating XML file is not as straightforward as DOM. After finishing this implementation the same test was run again. The application terminated within 10 minutes. Since SAX streams the data and triggers an event if certain keywords occur the data size which is kept in memory is compared to DOM in this case very small.

Since all the allowed character are known and the format of the XML file is very simple a third way of creating the XML file was tested. The file was created using the system functions `write()` and `read()` only. This implementation has the same complexity as the SAX implementation.

The test was run again several times using SAX and the system functions and Table 5.1 shows the final results:

TABLE 5.1: Parser Comparison

Used Model	Average Used Time
DOM	manually terminated after 30 minutes
SAX	7:53 minutes
System Functions	4:32 minutes

According to the results the XML creating is finally realised with the standard system functions.

5.5.2 Problems with XML

The log file may contain characters which are not allowed according to ISO 10646. If the XML file contains those characters the file is not well formed. That leads to exceptions during the transfer since the transfer is handled by a XML based server. Therefore those "illegal" characters have to be found. Deleting the unwanted characters is not a good option, since it would change the context of the log file entry. Thus, "illegal" characters are exchanged by the character "?". The user can recognise the exchange and if needed, can look up the original characters in the log file. Listing 5.3 shows the XML file `write_entry` function which is part of the `LogFileParser`.

LISTING 5.3: `write_entry` function

```python
def write_entry(self, tagname, content):
    '''
    This function inserts an entry into the xml file.

    tagname = tag name
    content = message between start and end tag
    '''
    #find all not allowed character
    bad_character = re.sub('[\x09\x0a\x0d\x20-\xd7]*', "", content)
    # replace each not allowed character with ”?”
    for i in range(len(bad_character)):
        if bad_character[i] == '\x00':
            # delete NUL character
            content = content.replace(bad_character[i], '')
        else:
            content = content.replace(bad_character[i], "?")

```

```
18      entry = "<%s>%s</%s>\n" % (tagname, content, tagname)
19      try:
20          self._client_log_file_fd.write(entry)
21          return 0
22      except IOError, e:
23          if self._verbose == 1:
24              print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(), e)
25          return -1
```

The actual work is done with Python's regular expression module from the standard library. In line 9 all the allowed characters are defined as hexadecimal numbers. The command `re.sub()` returns all not matched characters, in this case the "illegal" characters. Then each "illegal" character is exchanged. The character hexadecimal 00 is deleted, because it does not carrying any information.

5.6 Threads

Threads and their synchronisation was an important task to accomplish. Before the implementation is explained more detailed, some introducing words about threads.

A process can be seen as a running instance of an application. Each process has its own resources. A thread is a task within a process. The process can generate several simultaneously running threads. Contrary to processes, threads share resources *e.g.* memory. Therefore, threads can influence each other. Problems like deadlocks or race conditions can occur.

The client and server application work with threads. To synchronise the access of shared resources, *e.g.* the XML file, the mutex concept was implemented. Mutex stands for mutual exclusion. The gain mutual exclusion the thread has to acquire a "key". The "key" controls the access to the critical section. With critical section a code segment is referred where only one thread can be at a time, since shared resources or controlling variables are accessed there. If another thread wants to acquire the "key", the thread has to wait until the engaging thread releases the "key". The procedure of acquiring the "key" is atomic. One way of implementing mutex is the lock concept.

The lock can only be owned by one thread. A simple lock has two states, free or engaged. Python's standard library module threading contains such a mechanism. The lock provides the method `acquire()` and `release()`. Both methods are executed atomically [24].

The SQLite developer assert the software to be “threadsafe”. SQLite uses posix threads on Unix [25]. The gain thread safeness each thread has to call the `sqlite_open()` function. If several different processes try to access the database at the same time, where each single process has called automatically its own `sqlite_open()`, the process which comes second, receives an exception. The programmer can utilise the exceptions. In the project implementation, this function is only called during the initialising process. Later, only the access cursor is passed on to each thread. This way was chosen to avoid permanently initialising of the database. But this method requires synchronisation because if several threads use the same access cursor and try to access the database at the same time, the database behaviour is not predictable and it can lead to a software crash. Listing 5.4 shows the mutex implementation for the client, where the access to the database is synchronised.

Listing 5.4: Client Synchronisation Mechanism

```python
class Mutex:
    '''
    This class makes sure that only one client is writing into the database. This is
        necessary since sqlite is not thread safe within a process! Futhermore it
        provides the possiblity to synchronise threads accessing any critical
        sections within any other code segement.
    '''
    # database lock
    _db_locked = threading.Lock()
    # critical section lock
    _locked = threading.Lock()

    def __init__(self):
        '''
        Constructor
        '''
        self.writing = 0
        self._the_thread = 0

    def set_variable(self, threadus):
        '''
        set variable writing and the_thread
        '''
        Mutex._db_locked.acquire()
        if self.writing == 0:
            #set variable
            self.writing = 1
            self._the_thread = threadus
            Mutex._db_locked.release()
            return 0
        else:
            if (1 != self._the_thread.isAlive()):
                # if the thread, which set the variable is dead, reset variable
                self.writing = 0
            Mutex._db_locked.release()
            return -1

    def reset_variable(self):
        '''
        reset variable writing and the_thread
        '''
        Mutex._db_locked.acquire()
        self.writing = 0
        self._the_thread = 0
        Mutex._db_locked.release()

    def lock(self):
        '''
        This functions acquires the look.
        '''
        Mutex._locked.acquire()

    def release(self):
        '''
        This function releases the lock.
        '''
        Mutex._locked.release()
```

The database synchronisation consists of two functions. The function set_variable() sets the variable writing. Setting this variable is synchronised with the lock functions. Thus only one thread at a time is allowed to modify this variable. As a deadlock avoidance mechanism any thread can reset the variable with the function reset_variable(). Also another method to avoid deadlocks was implemented. Once a process discovers the variable is set, he checks if the thread which sets the variable is still alive. If this thread has already died the variable is reset automatically. To make this possible, each thread which sets the variable leaves his identity. The identity is deleted during the reset process.

The Mutex class also provides a lock for any other critical section that might occur. For example, assuming the client is fetching XML files from several server. Once the file is fetched it is temporarily saved on local disk. To avoid that the different threads interpenetrate each other by deleting the temporary files, each file has to have a unique name. The temporary file name is generated at run time. Therefore, each thread has to verify that the chosen name does not already exist. This verifying process is synchronised by the functions lock() and release() of the Mutex class. Only one thread at a time is able to determine a name for its temporary file.

At the server side the idea is to prevent the client from accessing the XML file, while the parser is still writing it and vice versa. This is realised with a similar mutex implementation as explained above.

5.7 Graphical User Interface

A graphical user interface can only be found in the Visualiser. With the parameter `-g` or `--graph` a pop-up window is generated. This pop-up window is an object of the class `Picture` which is derived from `Tkinter.Tk`. `Tkinter.Tk` generates the main window. Within the main window a frame is placed. Within the frame widgets such as buttons or listboxes are placed. But how are the widgets arranged in the frame?

`Tkinter` provides three different layout manager,

- `pack()`

  tries to arrange the widgets in a rectangle.

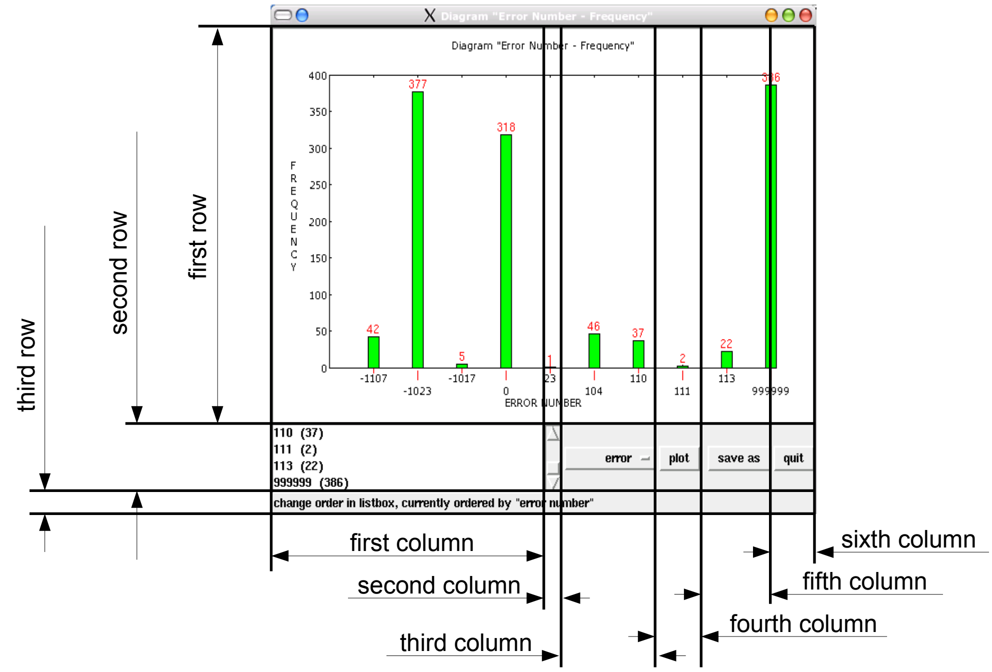
- `place()`

  allows to locate the widgets at absolute coordinates.

- `grid()`

  The geometry manager `grid()` manages the frame like a table with rows and columns.

For the project mainly the `grid()` manager was used, because this layout manager allows to place widgets very precise. The handling is straightforward and the layout is easy conceivably. To gain the same effect *e.g* with the `pack()` manager would require multiple nested frames. Figure 5.1 shows the grid design for the main window.



110 (37)
111 (2)
113 (22)
999999 (386)

change order in listbox, currently ordered by "error number"

FIGURE 5.1: Grid Layout

In the first row the graph itself is placed. The second row is used for user interactions. Finally the third row forms a status bar.

In the main window the user can see all errors and their total occurrence. The diagram can be influenced by the parameters as described in Table 4.23. The listbox in row 2 and column 1 displays the same information as the diagram itself. The user can select a listbox item and then press the button plot, which will give a more detailed diagram, *e.g.* done in the main windows it will generate a line diagram about a particular error as illustrated in Figure 5.2.

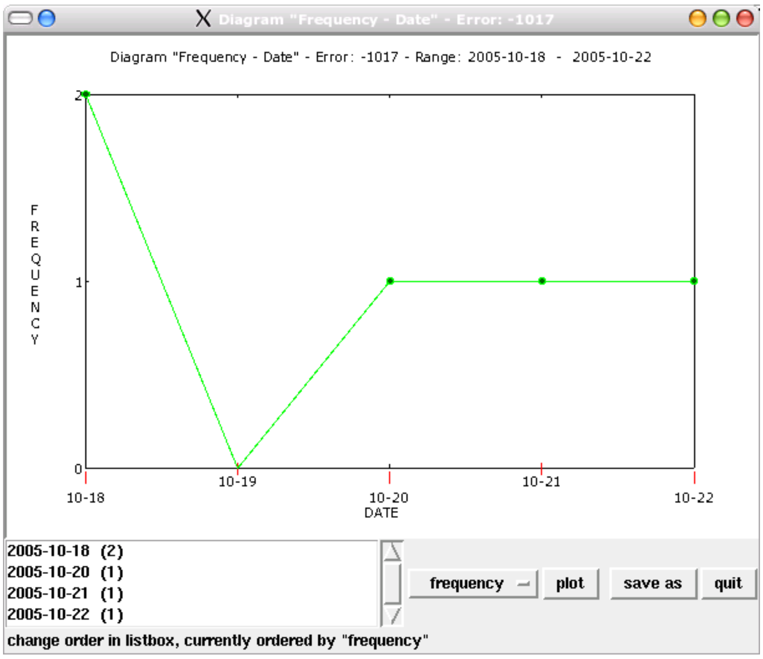


Diagram "Frequency - Date" - Error: -1017
Diagram "Frequency - Date" - Error: -1017 - Range: 2005-10-18 - 2005-10-22
FREQUENCY
2
1
0
10-18
10-19
10-20
10-21
10-22
DATE
frequency
plot
save as
quit

2005-10-18 (2)
2005-10-20 (1)
2005-10-21 (1)
2005-10-22 (1)

change order in listbox, currently ordered by "frequency"

FIGURE 5.2: Particular Error as Line Diagram

In the third column of the second row a drop down menu is placed. With this menu the items in the listbox can be ordered by error frequency or the corresponding value of the x-axis. The button “plot” triggers a new pop-up window with a new diagram from the listbox chosen item. The button “save as” is self-explanatory and triggers a dialog, where the location and name of the postscript file can be chosen. This dialog disables all other buttons in any other open window of the application. The user has to close the dialog first to carry on interacting with all the other windows. The button “quit” triggers a message box, which verifies the user wish to close the window. Again, first the message box has to be closed to interact with any other window of the application.

The implemented status bar supports the usability by displaying a short description. The status bar appears immediately after the mouse hovers over a widget. For a frequent user those short informations are sufficient. If the mouse is placed still on a widget for 3 seconds a tool tip occurs. The tool tips explains the usage of the widgets and are meant to support new users in the first place.

The `Picture` class is written in that way, that it can be used for all needed graphs. If the user wants to get more detailed information, just another object of the `Picture` class with modified data is created. The database is queried only **once**, since this is the most time-consuming process. Just the data required for the next graph is past on to the new object. Hence the all graphs, apart from the main window, can be created fast. This is possible because the primary database query receives all the needed datasets.

Each window (`Picture` object) keeps track of all from this window created objects. This was implemented to shut down the application quickly. Assuming the user opens ten windows, it would be a bit inconvenient to click on each window to terminate the application. Therefore if a window is closed all the “child” windows are closed as well. In this example, closing the main window will also close all the other nine windows.

5.8 Further Usability Improvements

Console application are usually more difficult to handle than applications with an intuitive graphical user interface. To improve the usability of the applications an extended help was implemented for each application invokable with the parameter `-h` or `--help`. The help explains each parameter and if required, provides examples to explain the usage of the application.

A further improvement was made with colouring the console output for the Visualiser and Remote Controller. The colour is changed by using ANSI (American National Standards Institute) escape codes. This are sequences of ASCII (American Standard Code for Information Interchange) characters. The codes can be used to control cursor movements and display graphics as well as reassign keys [26] on text terminals. The sequence starts with the escape character followed by a left bracket followed by alphanumeric characters. The extract from the class `Colour` as shown in Listing 5.5 illustrates the sequences used in this project.

LISTING 5.5: ANSI Escape Codes

```
class Colour:
    '''
    This class uses the ANSI escape sequences to color the output !
    '''
    color = {"reset":"\x1b[0m",
            "bold":"\x1b[01m",
            "teal":"\x1b[36;06m",
            "turquoise":"\x1b[36;01m",
            "fuscia":"\x1b[35;01m",
            "purple":"\x1b[35;06m",
            "blue":"\x1b[34;01m",
            "darkblue":"\x1b[34;06m",
            "green":"\x1b[32;01m",
            "darkgreen":"\x1b[32;06m",
            "yellow":"\x1b[33;01m",
            "brown":"\x1b[33;06m",
            "red":"\x1b[31;01m",
            "darkred":"\x1b[31;06m"}

    def __init__(self):
        '''
        Constructor
        '''
        pass

    def green(self, text):
        '''
        dye green
        '''
        return self.color['green']+text+self.color['reset']
```

The colours are defined in the dictionary `color`. If a console output needs to be coloured, the corresponding function is called. This function deals with colouring the output and takes care of resetting the colour scheme.

6 Evaluation and Results

This chapter will present the gained results and some tests, made to verify the results. Tests were run periodically during the development phase locally and after that on several systems across the network to create a realistic test environment. Not every test, which was made is mentioned here, just an overview about the made software evaluation is given.

To gain an overview about all the applications which got developed during this project, Figure 6.1 shows the applications and their connection to each other.

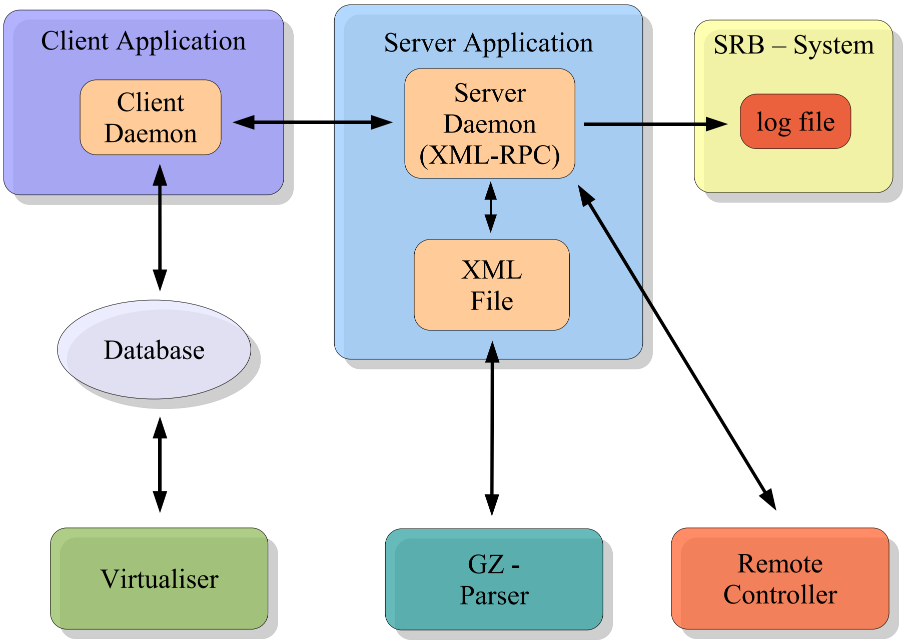


FIGURE 6.1: Application Overview

Each application was subject to extensive parameter evaluation and configuration file input evaluation, *e.g* detection of wrong IP addresses or missing files. For certain important modules little test applications were written. For example the database initialisation process for the database was developed as an extra module first. Only after this module passed all the tests, such as creating the database structure or inserting data, it was integrated into the main application. Further it was ensured that at least once the program was running through each loop. The progress was monitored with corresponding console output.

In general it can be said that the software was developed after the agile software development methodology. This includes thorough tests after the completion of certain development phases.

6.1 Server

It is important to carry out performance test. This will give insight about how good the program is handling system resources. The resources which are reasonable to be monitored for this project are

- CPU utilisation
- virtual memory usage
- disk space usage

The UNIX environment provides free tools to analyse and manage performance. Many tools offer information about the whole system only. But here the interest lies on certain processes. To view such information the tool `ps` can be very helpful. `ps` is a powerful tool that gives a snapshot of the current processes [27] and is able to display threads.

The server executes the most important task, the log file parsing. Hence the server was examined more closely concerning the performance. Table 6.1 shows the three system, that were used for testing the software. The properties are gathered from the systems itself, mainly from the `proc` folder.

TABLE 6.1: Test Systems

Property	Theodore	Rivers	escpc31
Processor Type	Intel(R) Pentium (R) M processor 1.60 GHz	Pentium III (coper-mine)	Intel (R) Pentium (R) 4 CPU 3.00 GHz
CPU	1,599.097 MHz	668,344 MHz	3,001.062 MHz
Cache	2048 KB	256 KB	1024 KB
RAM	515,064 KB	125,488 KB	513,264 KB
Linux	SuSE 9.2	SuSE 9.3	SuSE 9.3
Kernel	2.6.8-24.11-default	2.6.11.4-20.a-default	2.6.11.4-21.9smp
gcc	3.3.4	3.3.5	3.3.5

For testing purposes a relatively large log file with a size of 136,056 KB was produced. This file contained 1,706,925 log file entries. Each of the systems had to handle this file with

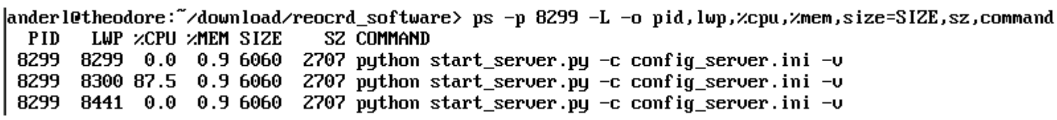
1. none keyword specified
2. one keyword specified
3. three keywords specified
4. eight keywords specified

Each single test was executed 20 times. Table 6.2 presents the average execution times. Also, it was tried not to occupy the systems with unnecessary tasks to receive comparable results. For the time measurement the Python standard library function `time()` was used.

TABLE 6.2: Test Results

System	parsing time (seconds)
keywords[0] - errors identified: 1,706,925 - XML file size: 395,012 KB	
Theodore	1,430.9015
Rivers	5,566.5937
escpc31	2,233.2219
keywords[1]:NOTICE:!status - errors identified: 569,315 - XML file size: 124,170 KB	
Theodore	1,180.5073
Rivers	4,912.1539
escpc31	1,935.6997
keywords[3]:NOTICE, findServerExec, Success - errors identified: 803 - XML file size: 173.409 KB	
Theodore	30.1293
Rivers	107.3109
escpc31	24.7377
keywords[8]:NOTICE, findServerExec, Success, svrCheckAuth, srbServerMain, portalConnect, connectPort, svrConnectSvr - errors identified: 0 - XML file size: 77 Byte	
Theodore	61.9903
Rivers	202.1009
escpc31	40.4878

The results differ according to the system properties. The parsing thread used most of the CPU capacities, in average 95 %. The thread is not sleeping during the parsing process and is therefore not giving up his CPU usage. The other threads did not utilise the CPU according to `ps`. `ps` only displays the avarage CPU utilisation of each process. But it proofs that the sleeping process is not using CPU capacities. This was expected and therefore, implemented this way. Depending on the system properties it takes different amounts of time, but none of the systems failed to work up the large log file. Figure 6.2 shows a `ps` measurement during a parsing activity. For each log file entry, additional information as well as the log file entry itself are saved in the XML file. Hence the XML became very large for the none keyword test.



```
anderl@theodore:~/download/reocrd_software> ps -p 8299 -L -o pid,lwp,%cpu,%mem,size=SIZE,sz,command
  PID   LWP %CPU %MEM SIZE    SZ COMMAND
 8299  8299  0.0  0.9 6060  2707 python start_server.py -c config_server.ini -v
 8299  8300 87.5  0.9 6060  2707 python start_server.py -c config_server.ini -v
 8299  8441  0.0  0.9 6060  2707 python start_server.py -c config_server.ini -v
```

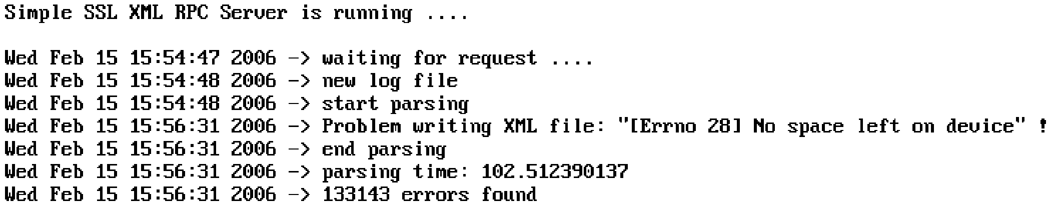
FIGURE 6.2: ps Output Server

The process has the PID 8299 identified beforehand with the command `ps -x`. The executed command

```
ps -p 8299 -L -o pid,lwp,%cpu,%mem,size=Size,sz,command
```

displays three threads associated with the process 8299 (PID). `LWP` stands for light weight process. The parsing thread has the number 8300 (LWP). The main thread has the same thread number as the process ID. As it was implemented, the parsing thread was created after the main thread. `Size` indicates the total size of the process in virtual memory, including all mapped files and devices, in kilobyte units [27]. The `ps` output shows that the server application uses very little memory compared to what large files the application is handling. It is also visible, that the thread with number 8441 (LWP) is serving a connected client and was **not** created immediately after the the parsing thread.

The test with such large files caused local disk memory problems due to the huge XML file creation on the test system “Theodore”. The application detected that successfully and informed the user (Figure 6.3).

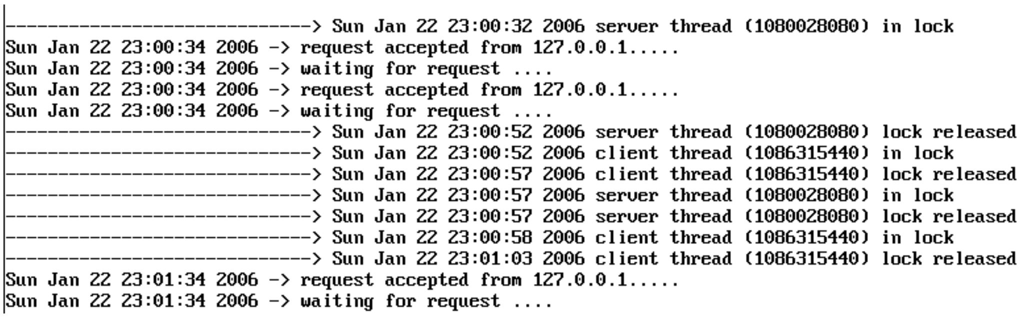


```
Simple SSL XML RPC Server is running ....

Wed Feb 15 15:54:47 2006 -> waiting for request ....
Wed Feb 15 15:54:48 2006 -> new log file
Wed Feb 15 15:54:48 2006 -> start parsing
Wed Feb 15 15:56:31 2006 -> Problem writing XML file: "[Errno 28] No space left on device" !
Wed Feb 15 15:56:31 2006 -> end parsing
Wed Feb 15 15:56:31 2006 -> parsing time: 102.512390137
Wed Feb 15 15:56:31 2006 -> 133143 errors found
```

FIGURE 6.3: Local Disk Space Problem

The parsing thread stopped the parsing process. But the main thread and therefore the application did not stop its work. The user can no free disk space and the parser will continue exactly where he stopped at the next parsing period.

The mutex mechanism was tested as well. To make the test results visible the `Mutex` class was temporarily extended with print instructions. Figure 6.4 shows the results.



```
-----------------------------> Sun Jan 22 23:00:32 2006 server thread (1080028080) in lock
Sun Jan 22 23:00:34 2006 -> request accepted from 127.0.0.1.....
Sun Jan 22 23:00:34 2006 -> waiting for request ....
Sun Jan 22 23:00:34 2006 -> request accepted from 127.0.0.1.....
Sun Jan 22 23:00:34 2006 -> waiting for request ....
-----------------------------> Sun Jan 22 23:00:52 2006 server thread (1080028080) lock released
-----------------------------> Sun Jan 22 23:00:52 2006 client thread (1086315440) in lock
-----------------------------> Sun Jan 22 23:00:57 2006 client thread (1086315440) lock released
-----------------------------> Sun Jan 22 23:00:57 2006 server thread (1080028080) in lock
-----------------------------> Sun Jan 22 23:00:57 2006 server thread (1080028080) lock released
-----------------------------> Sun Jan 22 23:00:58 2006 client thread (1086315440) in lock
-----------------------------> Sun Jan 22 23:01:03 2006 client thread (1086315440) lock released
Sun Jan 22 23:01:34 2006 -> request accepted from 127.0.0.1.....
Sun Jan 22 23:01:34 2006 -> waiting for request ....
```

FIGURE 6.4: Mutex Test

A server thread is entering the lock. While the thread engages the lock, two clients are connecting to the server. The client threads then try to access the lock, because they have to ensure the parser in not writing the XML file. Only after the server thread released the lock a client was able to enter. The in Figure 6.4 displayed sequence of different threads entering and leaving the lock proves, that the mutex mechanism works as designed and implemented.

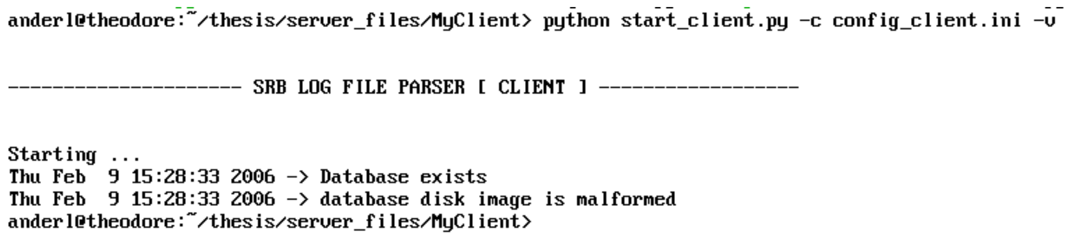
As for the server it can be said, that this application is very reliable. Mutex mechanism and deadlock avoidance mechanism make the server a highly stable application. The `ps` profiling of the server application confirmed that the server uses the memory efficient. Misconfiguration of the keyword file such as inserting none keyword can produce huge XML files.

6.2 Client

For the client the database is the most important issue. The database is only a file. With a file it is easy to tamper. Therefore before the client takes up its real work it checks if the database file is existing and if

- all database tables exist
- the error, host, project_host, project tables contain data
- the database file structure is intact

The result is reported to the user. In case of any anomaly the application terminates. The feature to detect database corruption became necessary due to the simplicity of manipulation of the database file. For testing purposes parts of the database file got manually deleted. Figure 6.5 displays the reaction of the application.

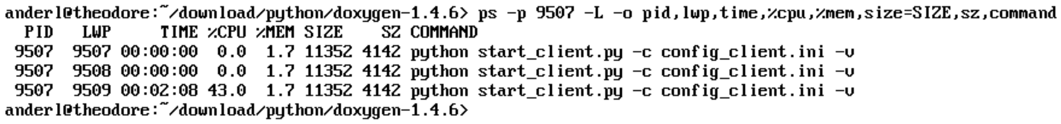


```
anderl@theodore:~/thesis/server_files/MyClient> python start_client.py -c config_client.ini -v

-------------------- SRB LOG FILE PARSER [ CLIENT ] -----------------


Starting ...
Thu Feb  9 15:28:33 2006 -> Database exists
Thu Feb  9 15:28:33 2006 -> database disk image is malformed
anderl@theodore:~/thesis/server_files/MyClient>
```

FIGURE 6.5: Database Corruption Detection

The database anomaly was detected. Therefore the user is notified that the database file structure is defective.

Furthermore, large XML files could successfully be transferred through the Internet. The database actualisation process was done without any interruption. Figure 6.6 shows the `ps` output for the client while inserting gathered information into the database.



```
anderl@theodore:~/download/python/doxygen-1.4.6> ps -p 9507 -L -o pid,lwp,time,%cpu,%mem,size=SIZE,sz,command
  PID   LWP     TIME %CPU %MEM SIZE    SZ COMMAND
 9507  9507 00:00:00  0.0  1.7 11352 4142 python start_client.py -c config_client.ini -v
 9507  9508 00:00:00  0.0  1.7 11352 4142 python start_client.py -c config_client.ini -v
 9507  9509 00:02:08 43.0  1.7 11352 4142 python start_client.py -c config_client.ini -v
anderl@theodore:~/download/python/doxygen-1.4.6>
```

FIGURE 6.6: `ps` Output Client

Three threads are visible. The main thread (LWP 9507) and the manager thread (LWP 9508) are currently sleeping since no CPU is used. The thread 9509 (LWP) is handling the database updating.

For the client similar performance test as made with the server were executed. The client application uses more virtual memory as the server but is still using the memory efficient. The database updating uses approximately 50% of the CPU and database initialising process uses as much as possible of the CPU capacity. Large files were handled without problems what make the client also a reliable application.

6.3 Other Applications

The Virtualiser queries the database. With different parameters the query can be specified. All parameters, also in combination, were tested successfully. Further, several instances were run at the same time to test the thread safeness of the database as well as the ability of the application to wait a certain time until the database is accessible again. None of the applications as well as the database failed. The tests were successfully completed.

The graphical user interface was subject to following major tests:

- window resizable
- graph savable as postscript file
- postscript file is readable

- buttons work according to the specified task
- quit button activation triggers new window and disables every other button in every other window
- the zoom was executed, missing data got inserted with the value 0

All test were successfully executed.

For the Remote Controller as a console application only the parameter, also in combination, got tested extensively. It was also tested if the collaboration with server worked reliable and the server fulfills the given task successfully.

The GZ Parser uses the same parsing module as the server. Therefore the parsing process is just as stable and reliable. The created XML file placement in the specified folder was accomplished.

To summarise, it can be said that all applications work reliable. Occurring problems are reported to the user with an appropriate advice.

7 Conclusion

This chapter is used to summarise the dissertation. Further, the original ideas are compared with the results and the achievements of the project presented.

7.1 Summary

This project was concerned with the development of tools to monitor the grid data management system Storage Resource Broker, originally developed by the SDSC in California (United States). An overview about the SRB was given in the beginning of this dissertation. The SRB systems work across the network. Therefore, some for this project developed tools are able to communicate through the network. Basic network technologies, Internet security issues and other fundamentals in conjunction with this project were explained for better understanding. Based on the client-server-architecture two main applications were developed. The SRB log file evaluation was the most important task to accomplish. The server is handling the SRB log file parsing. The parsing is configurable with external files. The wanted informations are saved into an XML file. This file is fetched from the client. The client saves the gathered and already structured information in a database. A client is able to observe several servers at the same time.

To complete the set of tools and to provide more usability three additional application were designed and developed. The Visualiser presents the database content in a clear way to the user. The user is able to specified the database query with parameters. Additionally a graph can be displayed. The graph shows the error frequency in general and if requested error frequency over a time period which is zoomable up to one day. Through the graphical user interface it is easily possible to save the graph as postscript file.

The application Remote Controller can be used to influence the parsing behaviour of the server remotely. The GZ Parser was developed to work up older SRB log files, which are already compressed to `*.gz` files.

7.2 Achievements

The original idea to develop monitoring tools is fully accomplished. The client-server-application is highly configurable in many aspects. Tests proved the client and server are working reliable and stable, even as daemons in the background.

The Virtualiser is a tool to envision the information, client and server have provided. The virtualisation process is also very flexible. Next to a coloured console output, a savable graph can be generated. This tool provides all necessary instruments to gain fast a wide overview about the SRB servers regarding occurring errors.

The SRB administrator does not have to be locally present to adjust the parsing parameters. With the Remoter Controller he has the possibility to influence the parsing process at any server, at any time, from any location. The tool saves therefore valuable time and provides flexibly for the SRB administrator.

The project requirements are met. The provided tools will support the SRB administration and ease the SRB system evaluation by identifying fast and easy problems within the SRB system. The developed applications are not only restricted to the SRB system. Any other system with a text log file can be monitored.

8 Future Prospects

Software products are never finished. There will always be issues which are missing or could be done in a different way. This chapter just points out a few aspects for future developments.

For this project Python Version 2.2.3 was used to be compatible with already existing software products. But analyses and development showed that a higher Python version would have eased the implementation process. Therefore, to be able to use already existing Python modules such as the “pyparser” introduced in chapter 3.1 an up-to-date version of Python is recommended for further software developments. The current version is Python 2.4.2 [13].

The graphical user interface fulfils the requirements. The implementation was done with the standard library only. For further extensively diagrams and graphs the implementation process might become very complex. *The `matplotlib` is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms* [28] and would serve this purpose well.

At the moment a connection can only be established to the server without a firewall or the firewall has to be configured especially. A firewall restricts the access from the outside network to network resources within a private (inside) network. To avoid opening the firewall and still to be able to run the software, HTTP tunnelling could be a solution, since HTTP connections are usually allowed. Most of companies, institutes, or universities use proxy servers as part of their firewall. A proxy server is computer system which handles the data transfer from the outside to the inside network. A application can send a request to the proxy server. The proxy server will then execute the request and send the answer back to the application. The developed software could be extended, that the connections are made using the proxy server.

References

[1] Grid-Computing. http://de.wikipedia.org/wiki/Gridcomputing.

[2] Council for the Central Laboratory of the Research Councils (CCLRC). http://www.cclrc.ac.uk/Activity/WhoWeAre.

[3] CCLRC e-Science Centre. http://www.e-science.clrc.ac.uk/web.

[4] SDSC Storage Resource Broker. http://www.sdsc.edu/srb.

[5] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, Ch. Cowart, B. Zhu, S.-Y. Chen, and R. Olschanowsky. Storage Resource Broker - Managing Distributed Data in a Grid. Technical report, San Diego Supercomputer Center (SDSC), University of California at San Diego.

[6] C. Hunt. *TCP/IP Network Administration*. O'Reilly, second edition, December 1997.

[7] AppleTalk. http://de.wikipedia.org/wiki/Apple_Talk.

[8] R Fielding, J. Gettys, J Mogul, H. Frystyk, L Masinter, P Leach, and T Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. http://www.ietf.org/rfc/rfc2616.txt.

[9] J. Viega, M. Messier, and P. Chandra. *Network Security with OpenSSL*. O'Reilly Media Inc., first edition, 2002.

[10] Secure Socket Layer. http://www.windowsecurity.com/articles/Secure_Socket_Layer.html.

[11] N. Walsh. A Technical Introduction to XML. http://www.xml.com/pub/a/98/10/guide0.html?page=2#AEN63, 1998.

[12] C.J. Date. *Introduction to Database Systems*. Addison-Wesley, eighth edition edition, 2004.

[13] Python. http://www.python.org.

[14] P. McGuire. pyparsing – an object-oriented approach to text processing in Python. http://pyparsing.sourceforge.net/.

[15] Unix Programming Frequently Asked Questions - Process Control. http://www.erlenstar.demon.co.uk/unix/faq_2.html, September 2000.

[16] OpenSSL. http://www.openssl.org, October 2005.

[17] M. Sjögren. Python OpenSSL Manual. Technical report, Release 0.6. http://pyopenssl.sourceforge.net/pyOpenSSL.html/pyOpenSSL.html.

[18] SWIG - Executive Summary. http://swig.sourceforge.net/exec.html, 2004.

[19] M2Crypto = Python + OpenSSL + SWIG. http://sandbox.rulemaker.net/ngps/m2.

[20] About SQLite. http://www.sqlite.org.

[21] J. E. Grayson. Python and Tkinter Programming. Manning.

[22] Manifesto for Agile Software Development. http://www.agilemanifesto.org, October 2005.

[23] Doxygen. http://www.stack.nl/ dimitri/doxygen, January 2005.

[24] G. van Rossum. Lock Objects - Python Library Reference. Technical report, PythonLabs, May 2003. http://www.python.org/doc/2.2.3/lib/lock-objects.html.

[25] SQLite Threading. http://sandbox.rulemaker.net/ngps/149.

[26] ANSI CODES. http://rrbrandt.dyndns.org:60000/docs/tut/redes/ansi.php.

[27] B. Lankester, M. K. Johnson, and R. Sladkey. ps Linux User's Manual. Technical report, July 2004.

[28] Matplotlib. http://matplotlib.sourceforge.net, November 2005.

[29] A. Rajasekar, M. Wan, and R. Moore. MySRB & SRB - Components of a Data Grid. Technical report, San Diego Supercomputer Center, University of California at San Diego.

[30] AA Verstraete. Data Communications Protocols In-Depth. http://www.smeal.psu.edu/misweb/datacomm/ID/ID_PROTO.BAK, 1998.

[31] Information Sciences Institute. TRANSMISSION CONTROL PROTOCOL (RFC 793). Technical report, Information Sciences Institute, University of Southern California, California, 1981.

[32] Information Sciences Institute. INTERNET PROTOCOL (RFC 791). Technical report, Information Sciences Institute, University of Southern California, California, 1981.

[33] J. Plate and J. Holzmann. Sichere Protokolle. http://www.inf-wiss.uni-konstanz.de/CURR/summer00/ec/sicher.html.

[34] IETF. Transport Layer Security (TLS). http://www.ietf.org/html.charters/tls-charter.html, 2005.

[35] DOM - Document Object Model. http://www.w3.org/TR/REC-DOM-Level-1/introduction.html.

[36] Official Website for SAX. http://www.saxproject.org/.

[37] H. Herold. *Linux/ Unix - Programmierung*. Addison-Wesley-Longman, 4. revised edition, 2002.

[38] R. Fischbach. Beschraenkung aufs Wesentliche. *iX - Magazin fuer professionelle Informationstechnik*, 11, 1999. http://www.heise.de/ix/artikel/1999/11/184/.

[39] About pysqlite. http://initd.org/tracker/pysqlite/wiki/About.

[40] A. Martelli, A. Martelli Ravenscroft, and D. Ascher. *Python Cookbook*. O'Reilly, second edition, March 2005.

[41] M. Ascher, D.and Lutz. *Learning Python*. O'Reilly, second edition, December 2003.

[42] K. Günther. *LaTeX GE-PACKT*. mitp, first edition, 2002.

[43] M. Weigend. *Python GE-PACKT*. mitp, second edition, 2005.

[44] K. Günther. *Linux GE-PACKT*. mitp, second edition, 2002.

[45] J Klensin. Simple Mail Transfer Protocol. Technical report, AT&T Laboratories, 2001.

[46] J. Goebel, A. Hasan, and F. S. Tehrani. *The Book of Python - From the Tip of the Tongue to the End of the Tale*. No Starch Press, expected in June 2006.

[47] J. W. Shipman. Tkinter reference: a GUI for Python. Technical report, New Mexico Tech Computer Center, August 2005. www.nmt.edu/tcc.

[48] fabFORCE.net. DB Designer 4. http://fabforce.net/dbdesigner4.

[49] ActiveState. Komodo. http://www.activestate.com.

[50] B. Dufour. A Comprehensive Introduction to Python Programming. Technical report, McGill Univeristy.

[51] SRB Workshop. http://www.sdsc.edu/srb/Workshop, February 2006.

[52] K. Schwaber. SCRUM - It's About Common Sense. http://www.controlchaos.com, December 2005.

[53] Informal Language Comparison Chart(s). http://www.smallscript.org.

[54] E. Raymond. Why Python? *LINUX JOURNAL*, 2000.

[55] MySQL. http://www.mysql.de.

[56] PostgreSQL. http://www.postgresql.org.

Appendix A

Development Environment

For the design, development and documentation following software were used during this project:

- DB Designer 4.0.4.9 Beta [48]
- SuSE Linux 9.2
- Komodo 3.2 Trail [49]
- gcc 3.3.4
- egrep 2.5.1
- GNU Awk 3.1.4
- GNU bash 3.00.0(1)
- Python 2.2.3 [13]
- $\LaTeX 2_\varepsilon$
- M2Crypto 0.13 [19]
- SQLite 2.8.16 [20]
- pysqlite 1.0.1 [39]
- $\TeX$*nicCenter* 1 *Beta* 6.31[1]
- MiK$\TeX$ 2.4[2]
- Umbrello
- doxygen 1.4.6[3]

[1] http://www.toolscenter.org

[2] http://www.miktex.org

[3] http://www.stack.nl/d̃imitri/doxygen/

Table 6.1 shows the systems which were used for testing the software. “Theodore” was the development system.

TABLE A.1: Test Systems

Property	**Theodore**	**Rivers**	**escpc31**
		Hardware Properties	
Processor Type	Intel(R) Pentium (R) M processor 1.60 GHz	Pentium III (coper-mine)	Intel (R) Pentium (R) 4 CPU 3.00 GHz
CPU	1,599.097 MHz	668,344 MHz	3,001.062 MHz
Cache	2048 KB	256 KB	1024 KB
RAM	515,064 KB	125,488 KB	513,264 KB
		Software Properties	
Linux	SuSE 9.2	SuSE 9.3	SuSE 9.3
Kernel	2.6.8-24.11-default	2.6.11.4-20.a-default	2.6.11.4-21.9smp
gcc	3.3.4	3.3.5	3.3.5

Appendix B

Detailed Class Diagrams

B.1 Remote Controller

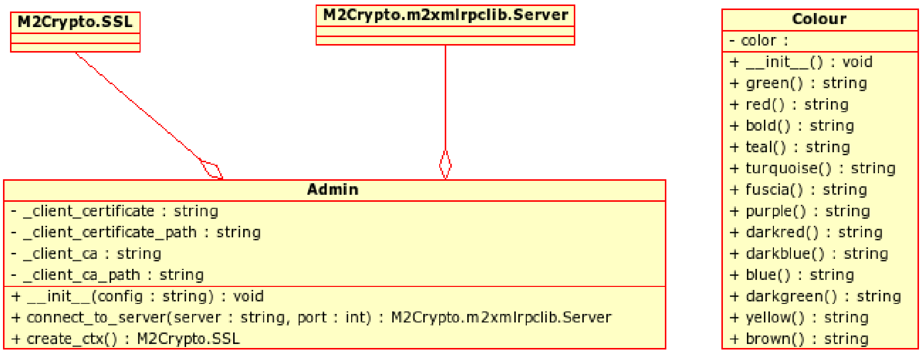


FIGURE B.1: Remote Controller Class Diagram

B.2 Server

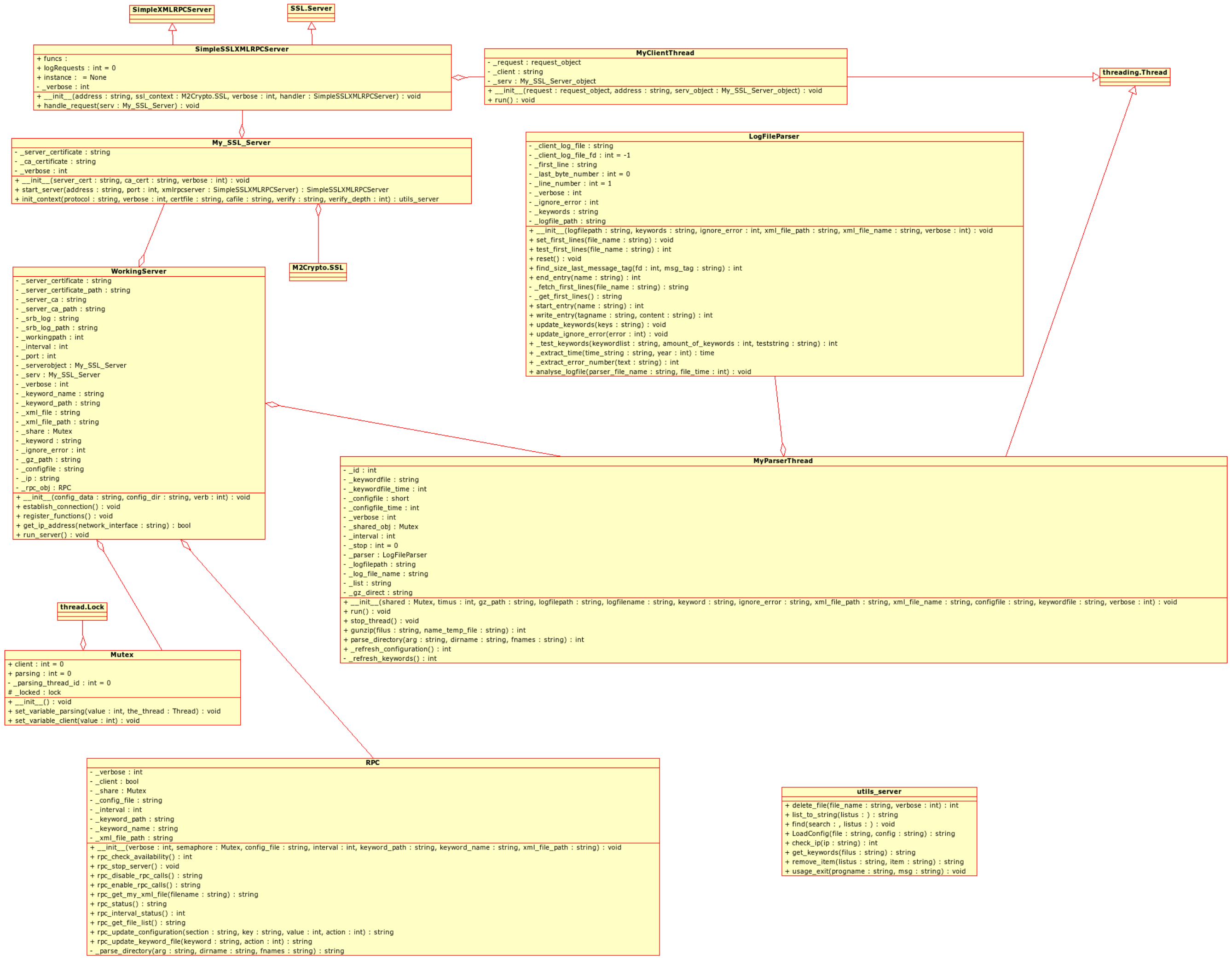

SimpleXMLRPCServer
SSL.Server
SimpleSSLXMLRPCServer
+ funcs :
+ logRequests : int = 0
+ instance : = None
- _verbose : int
+ __init__(address : string, ssl_context : M2Crypto.SSL, verbose : int, handler : SimpleSSLXMLRPCServer) : void
+ handle_request(serv : My_SSL_Server) : void
MyClientThread
- _request : request_object
- _client : string
- _serv : My_SSL_Server_object
+ __init__(request : request_object, address : string, serv_object : My_SSL_Server_object) : void
+ run() : void
threading.Thread
My_SSL_Server
- _server_certificate : string
- _ca_certificate : string
- _verbose : int
+ __init__(server_cert : string, ca_cert : string, verbose : int) : void
+ start_server(address : string, port : int, xmlrpcserver : SimpleSSLXMLRPCServer) : SimpleSSLXMLRPCServer
+ init_context(protocol : string, verbose : int, certfile : string, cafile : string, verify : string, verify_depth : int) : utils_server
LogFileParser
- _client_log_file : string
- _client_log_file_fd : int = -1
- _first_line : string
- _last_byte_number : int = 0
- _line_number : int = 1
- _verbose : int
- _ignore_error : int
- _keywords : string
- _logfile_path : string
+ __init__(logfilepath : string, keywords : string, ignore_error : int, xml_file_path : string, xml_file_name : string, verbose : int) : void
+ set_first_lines(file_name : string) : void
+ test_first_lines(file_name : string) : int
+ reset() : void
+ find_size_last_message_tag(fd : int, msg_tag : string) : int
+ end_entry(name : string) : int
- _fetch_first_lines(file_name : string) : string
- _get_first_lines() : string
+ start_entry(name : string) : int
+ write_entry(tagname : string, content : string) : int
+ update_keywords(keys : string) : void
+ update_ignore_error(error : int) : void
+ _test_keywords(keywordlist : string, amount_of_keywords : int, teststring : string) : int
+ _extract_time(time_string : string, year : int) : time
+ _extract_error_number(text : string) : int
+ analyse_logfile(parser_file_name : string, file_time : int) : void
WorkingServer
- _server_certificate : string
- _server_certificate_path : string
- _server_ca : string
- _server_ca_path : string
- _srb_log : string
- _srb_log_path : string
- _workingpath : int
- _interval : int
- _port : int
- _serverobject : My_SSL_Server
- _serv : My_SSL_Server
- _verbose : int
- _keyword_name : string
- _keyword_path : string
- _xml_file : string
- _xml_file_path : string
- _share : Mutex
- _keyword : string
- _ignore_error : int
- _gz_path : string
- _configfile : string
- _ip : string
- _rpc_obj : RPC
+ __init__(config_data : string, config_dir : string, verb : int) : void
+ establish_connection() : void
+ register_functions() : void
+ get_ip_address(network_interface : string) : bool
+ run_server() : void
M2Crypto.SSL
MyParserThread
- _id : int
- _keywordfile : string
- _keywordfile_time : int
- _configfile : short
- _configfile_time : int
- _verbose : int
- _shared_obj : Mutex
- _interval : int
- _stop : int = 0
- _parser : LogFileParser
- _logfilepath : string
- _log_file_name : string
- _list : string
- _gz_direct : string
+ __init__(shared : Mutex, timus : int, gz_path : string, logfilepath : string, logfilename : string, keyword : string, ignore_error : string, xml_file_path : string, xml_file_name : string, configfile : string, keywordfile : string, verbose : int) : void
+ run() : void
+ stop_thread() : void
+ gunzip(filus : string, name_temp_file : string) : int
+ parse_directory(arg : string, dirname : string, fnames : string) : int
+ _refresh_configuration() : int
- _refresh_keywords() : int
thread.Lock
Mutex
+ client : int = 0
+ parsing : int = 0
- _parsing_thread_id : int = 0
# _locked : lock
+ __init__() : void
+ set_variable_parsing(value : int, the_thread : Thread) : void
+ set_variable_client(value : int) : void
RPC
- _verbose : int
- _client : bool
- _share : Mutex
- _config_file : string
- _interval : int
- _keyword_path : string
- _keyword_name : string
- _xml_file_path : string
+ __init__(verbose : int, semaphore : Mutex, config_file : string, interval : int, keyword_path : string, keyword_name : string, xml_file_path : string) : void
+ rpc_check_availability() : int
+ rpc_stop_server() : void
+ rpc_disable_rpc_calls() : string
+ rpc_enable_rpc_calls() : string
+ rpc_get_my_xml_file(filename : string) : string
+ rpc_status() : string
+ rpc_interval_status() : int
+ rpc_get_file_list() : string
+ rpc_update_configuration(section : string, key : string, value : int, action : int) : string
+ rpc_update_keyword_file(keyword : string, action : int) : string
- _parse_directory(arg : string, dirname : string, fnames : string) : string
utils_server
+ delete_file(file_name : string, verbose : int) : int
+ list_to_string(listus :) : string
+ find(search : , listus :) : void
+ LoadConfig(file : string, config : string) : string
+ check_ip(ip : string) : int
+ get_keywords(filus : string) : string
+ remove_item(listus : string, item : string) : string
+ usage_exit(progname : string, msg : string) : void

FIGURE B.2: Server Class Diagram

B.3 Client

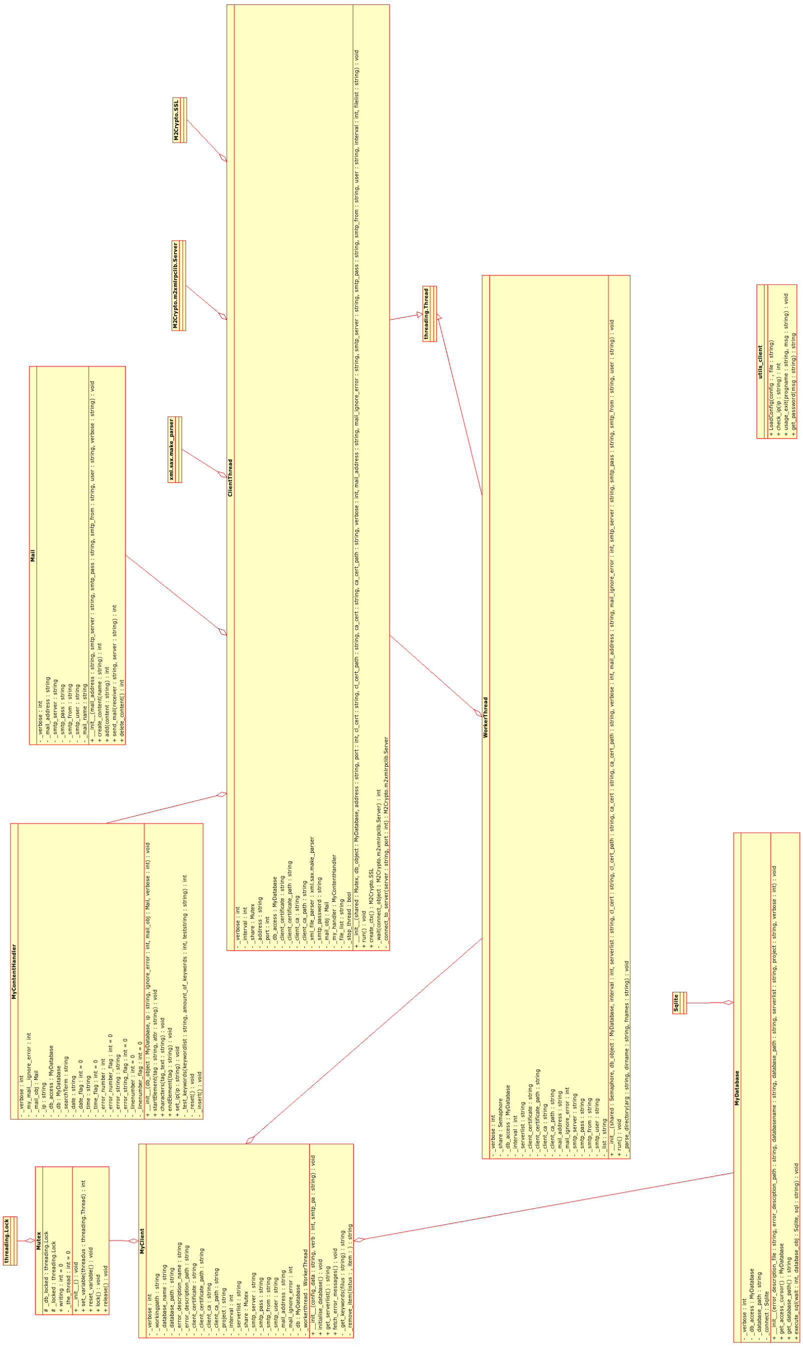


FIGURE B.3: Client Class Diagram

B.4 Virtualiser

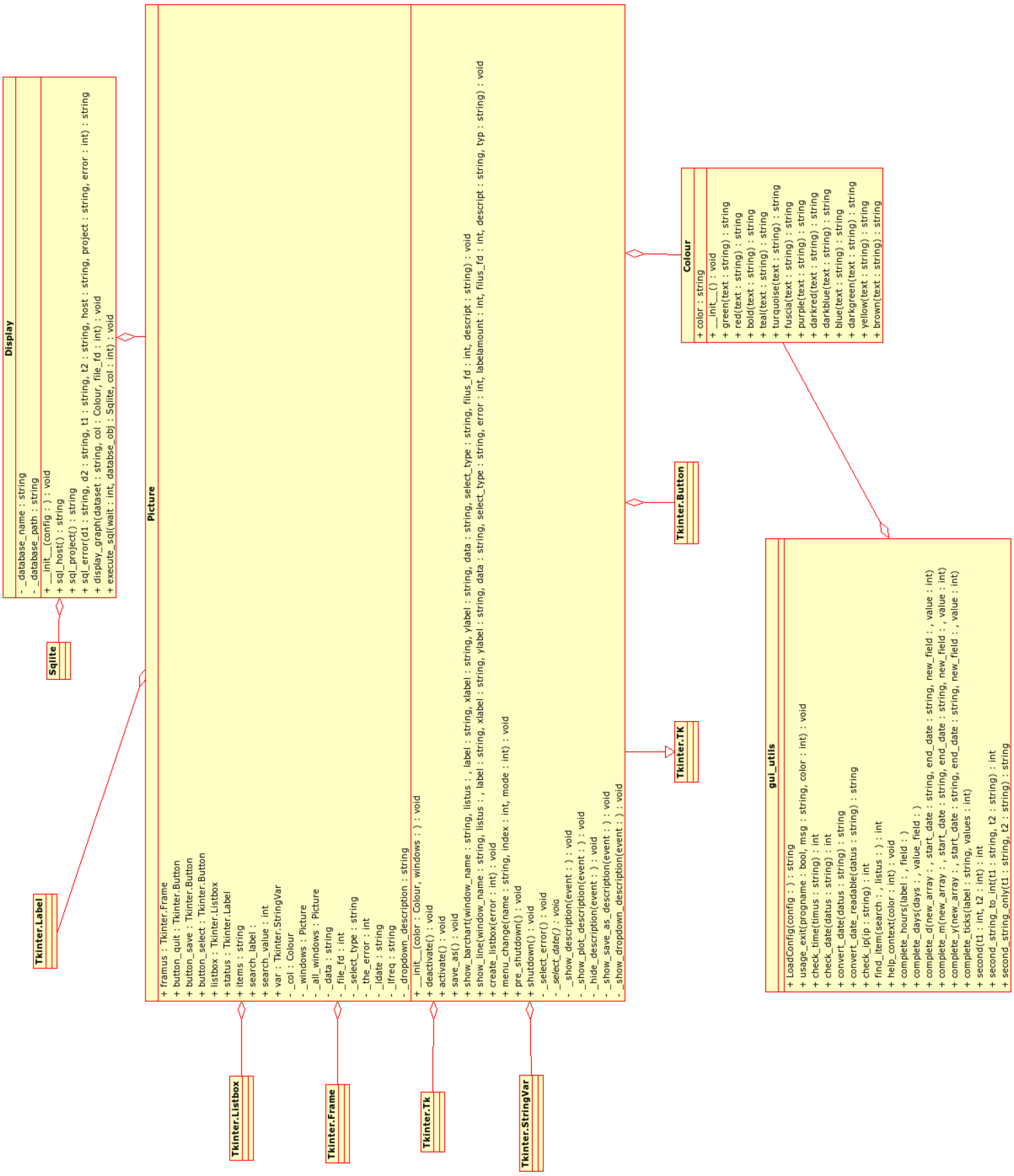

Display
- _database_name : string
- _database_path : string
+ __init__(config :) : void
+ sql_host() : string
+ sql_project() : string
+ sql_error(d1 : string, d2 : string, t1 : string, t2 : string, host : string, project : string, error : int) : string
+ display_graph(dataset : string, col : Colour, file_fd : int) : void
+ execute_sql(wait : int, databse_obj : Sqlite, col : int) : void
Sqlite
Tkinter.Label
Picture
+ framus : Tkinter.Frame
+ button_quit : Tkinter.Button
+ button_save : Tkinter.Button
+ button_select : Tkinter.Button
+ listbox : Tkinter.Listbox
+ status : Tkinter.Label
+ items : string
+ search_label :
+ search_value : int
+ var : Tkinter.StringVar
- _col : Colour
- _windows : Picture
- _all_windows : Picture
- _data : string
- _file_fd : int
- _select_type : string
- _the_error : int
- _ldate : string
- _lfreq : string
- _dropdown_description : string
+ __init__(color : Colour, windows :) : void
+ deactivate() : void
+ activate() : void
+ save_as() : void
+ show_barchart(window_name : string, listus : , label : string, xlabel : string, ylabel : string, data : string, select_type : string, filus_fd : int, descript : string) : void
+ show_line(window_name : string, listus : , label : string, xlabel : string, ylabel : string, data : string, select_type : string, error : int, labelamount : int, filus_fd : int, descript : string, typ : string) : void
+ create_listbox(error : int) : void
+ menu_change(name : string, index : int, mode : int) : void
+ pre_shutdown() : void
+ shutdown() : void
- _select_error() : void
- _select_date() : void
- _show_description(event :) : void
- _show_plot_description(event :) : void
- _hide_description(event :) : void
- _show_save_as_description(event :) : void
- _show_dropdown_description(event :) : void
Tkinter.Listbox
Tkinter.Frame
Tkinter.Tk
Tkinter.StringVar
Tkinter.TK
Tkinter.Button
Colour
+ color : string
+ __init__() : void
+ green(text : string) : string
+ red(text : string) : string
+ bold(text : string) : string
+ teal(text : string) : string
+ turquoise(text : string) : string
+ fuscia(text : string) : string
+ purple(text : string) : string
+ darkred(text : string) : string
+ darkblue(text : string) : string
+ blue(text : string) : string
+ darkgreen(text : string) : string
+ yellow(text : string) : string
+ brown(text : string) : string
gui_utils
+ LoadConfig(config :) : string
+ usage_exit(progname : bool, msg : string, color : int) : void
+ check_time(timus : string) : int
+ check_date(datus : string) : int
+ convert_date(datus : string) : string
+ convert_date_readable(datus : string) : string
+ check_ip(ip : string) : int
+ find_item(search : , listus :) : int
+ help_context(color : int) : void
+ complete_hours(label : , field :)
+ complete_days(days : , value_field :)
+ complete_d(new_array : , start_date : string, end_date : string, new_field : , value : int)
+ complete_m(new_array : , start_date : string, end_date : string, new_field : , value : int)
+ complete_y(new_array : , start_date : string, end_date : string, new_field : , value : int)
+ complete_ticks(label : string, values : int)
+ second(t1 : int, t2 : int) : int
+ second_string_to_int(t1 : string, t2 : string) : int
+ second_string_only(t1 : string, t2 : string) : string

FIGURE B.4: Virtualiser Class Diagram

Appendix C

Software User Manuals

C.1 Server

The SimpleSSLXMLRPCServer is parsing the SRB log file and is a console application, only controllable through parameters. The application requires Python 2.2.3 or higher, M2Crypto 0.13, bash, egrep, and awk. Table C.1 displays all available parameters.

TABLE C.1: Server Parameters

Parameter	Explanation
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]
-d or –daemon	daemonise the server

The server prints messages on the screen is `-v` is passed on. A message is printed if an event happens such as a client is connecting or the server is parsing. If `-v` and `-d` are passed, the messages are printed into a log file, which is placed in the same folder as the start script is located. The help can be seen with `-h`. This parameter disables all other passed parameters. The help contains basic examples too.

Before the server can be started with

```
python start_server.py -c configuration_file.ini [-v | -d | -h]
```

the configuration file has to be adjusted.

C.1.1 Configure the Server

Table C.2 explains all parts of the configurations file.

TABLE C.2: Configuration File Server

Parameter	Explanation
	Section Files
server_certificate	name of the server certificate file
server_ca	name of the ca file
srb_log	name of SRB log file
keyword	name of keyword file
	Section Path
path_server_certificate	path of the server certificate file
path_server_ca	path of the ca file
path_srb_log	path of SRB server log file
path_gz	path of SRB server gz log files (old log files)
path_keyword	path of keyword file
	Section Misc
minute	how often should the server parse the srb log file, *e.g.* 30 ⇒ means every 30 minutes
interface	network interface *e.g.* lo or eth0 or eth1
port	port on which the server is listening, default is 6000
ignore_error	some error numbers might not be interesting, so the errors should be ignored, *e.g.* 0, 3, 5 (comma separated list)

The server can be stopped with the bash script `stop_server.sh`.

C.1.2 Examples

For a better understand some simple examples are given:

1. `python start_server -c config_server.ini -v`
   run server within a console and print messages

2. `python start_server -c config_server.ini -v -d`
   run server as daemon, messages are written in log file

3. `python start_server -c config_server.ini -d`
   run server as daemon

C.2 Client

The client is fetching the preprocessed data from the client and inserts the information into a database. The application requires Python 2.2.3 or higher, sqlite 2.8.16, pysqlite 1.0.1, bash, egrep, and awk. This console application is controllable through the in Table C.3 displayed parameter.

TABLE C.3: Client Parameters

Parameter	Explanation
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]
-p or –smtp_password	activates mail notification sending
-d or –daemon	daemonize the client

The client prints messages on the screen is `-v` is passed on. A message is printed if an event happens such as a client is connecting to a server. If `-v` and `-d` are passed, the messages are printed into a log file, which is placed in the same folder as the start script is located. The help can be seen with `-h`. This parameter disables all other passed parameters. The help contains basic examples too. The mail notification can be activated with -p. This parameter invokes the application to require the password for the SMTP server. The connection to the SMTP server is tested. If the test fails, the client will terminate.

To be able to run the client with the command

```
python start_client.py -c configuration_file.ini [-v | -d | -p | -h]
```

the configuration file has to be adjusted first.

C.2.1 Configure the Client

Table C.4 explains all parts of the configuration file.

TABLE C.4: Configuration File Client

Parameter	**Explanation**
Parameter	**Explanation**
	Section Database
name	name of the database file
path	path of the database file
	Section Files
error_description	name of the error description file
client_certificate	name of the clients certificate file
client_ca	name of the ca file
	Section Path
path_error_description	path error description file
path_client_certificate	path client certificate
path_client_ca	path ca file
	Section Server
serverlist	the servers including the port, *e.g.* server1_IP:server1_port,server2_IP:server2_port
	Section Misc

Continued on next page

Table C.4 Configuration File Client - *continued from previous page*

Parameter	**Explanation**
minute	how often should the client fetch the XML file from server in minutes
	Section Project
name	the name of the project, at the moment only one project is possible
	Section Mail
smtp_server	SMTP server address
user	user name for the mail account
from	mail identification (where does the mail come from), please note that some mail server does not support own identifications
	Section Mail To
address_1	email address of the 1. person
file_keyword_1	file where keywords are defined
path_keyword_1	path of keyword file for 1. person
address_2	email address of the 2. person
file_keyword_2	file where keywords are defineend
path_keyword_2	path of keyword file for 2. person

The section *"Mail To"* can be extended to as many persons as needed. It is only important to follow the predefined pattern. The client can be stopped with the bash script `stop_client.sh`.

The client fetches the XML files from the server and saves the files temporary on local disk. If for some reason the XML processing is interrupted, within the next parsing period the client deals with the older files too.

C.2.2 Examples

For a better understand some simple examples are given:

1. `python start_client -c config_client.ini -v`
   run client within a console and print messages

2. `python start_client -c config_client.ini -v -d`
   run client as daemon, messages are written in log file

3. `python start_client -c config_client.ini -d`
   run client as daemon

4. `python start_client -c config_client.ini -p`
   run client within a console and activate mail notification

C.3 Virtualiser

The Visualiser can be used to present the database content. This application provides instruments to gain precise, user specified data. The application requires Python 2.2.3 or higher, sqlite 2.8.16, and pysqlite 1.0.1. This console application is controllable through parameters, listed in Table C.5.

TABLE C.5: Virtualiser Parameters

Parameter	Explanation
	general parameters
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]
-g or –graph	show output additionally as a diagram
–nocolor	no coloured console output
–file <string>	dump output into a file (file name has to be given)
	database commands
–sql_host	show all hosts
–sql_project	show all projects
–sql_error	show errors (additional parameters possible)
–sql_error_freq	show only frequency of errors (additional parameters possible)

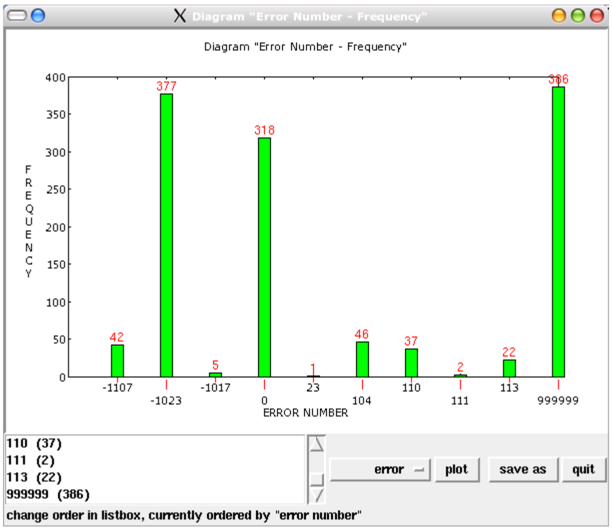
Continued on next page

Table C.5 Virtualiser Parameters - *continued from previous page*

Parameter	Explanation
	additional parameters
–start_date <date>	start date (*e.g.* 23.12.2005)
–end_date <date>	end date (*e.g.* 23.01.2006)
–start_time <time>	start time (*e.g.* 23:12:19)
–end_time <time>	end time (*e.g.* 23:12:59)
–ip <ip>	host IP (*e.g.* 127.0.0.1)
–project <string>	specify a certain project
–error <int,int...>	specify a certain error (comma separated list)

The database commands can only be used once at a time. The additional commands can be combined to define a certain interesting range or error, respectively. A graph (`-g`) can only be produced in conjunction with the parameter `--sql_error`. The GUI usage is straight forward and self-explanatory. Individual widgets are explained in the status bar or if the mouse hovers for more than 3 seconds on a widget with tool tips, which are faded in. The created graph can be saved as a postscript file. A dialog is leading through the saving process. As long as an additional dialog or a message box is not closed any other button within all windows of the application are disabled. Please close these additional windows first to enable those buttons again. If the graph does not deliver the needed information, please close the windows and specify the parameters accordingly to gain the needed information.

Figure C.1 depicts the main window.



110 (37)
111 (2)
113 (22)
999999 (386)

change order in listbox, currently ordered by "error number"

FIGURE C.1: Main Window

The bar chart graph shows all occurring errors. The error number is used as x-axis description. The error frequency is displays on top of each bar. The list box contains the same information, error number and in brackets the frequency. If more information about a certain error required, a error can be selected within the listbox. The button “plot” will then generate a new window like displayed in figure C.2. The order within the listbox can be changed with the dropdown menu. The listbox can be ordered error number or error frequency.

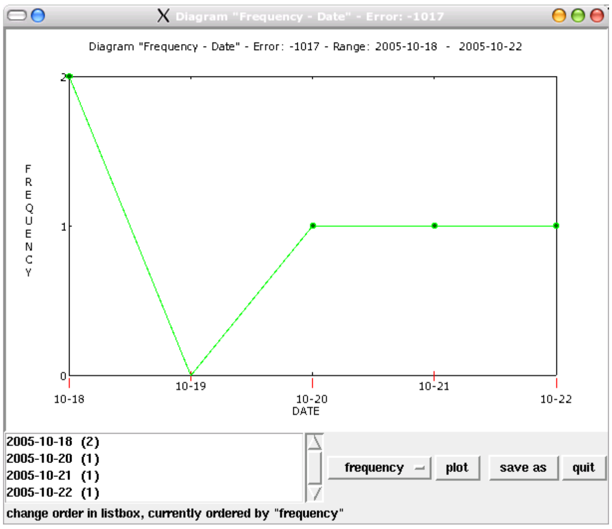


Diagram "Frequency - Date" - Error: -1017

Diagram "Frequency - Date" - Error: -1017 - Range: 2005-10-18 - 2005-10-22

2005-10-18 (2)
2005-10-20 (1)
2005-10-21 (1)
2005-10-22 (1)

change order in listbox, currently ordered by "frequency"

FIGURE C.2: Error Window I

The design of the window is similar to the main window. Displayed is line diagram which shows the error frequency over a certain period of time (days). In the listbox only those days appear where an error was reported. A day can now be chosen and Figure C.3 presents the final window. Multiple windows are possible.

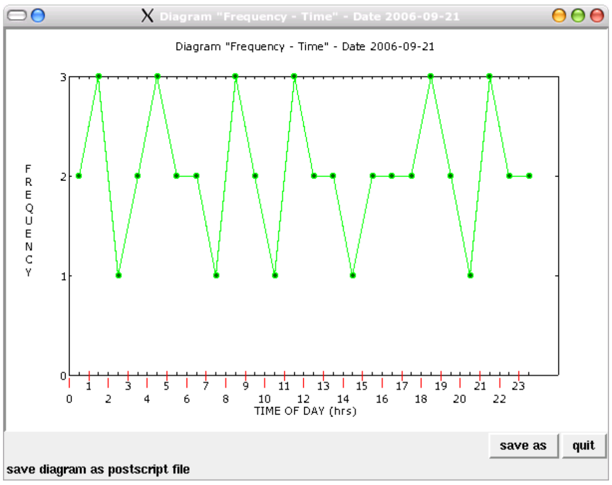


Diagram "Frequency - Time" - Date 2006-09-21

Diagram "Frequency - Time" - Date 2006-09-21

save diagram as postscript file

FIGURE C.3: Error Window II

The design is again similar. The window shows a line diagram of one particular error on one particular day. There is no listbox, because another zoom is not possible anymore. Multiple windows are possible.

C.3.1 Examples

For a better understand some simple examples are given:

1. `python gui.py -c config_gui.ini --sql_project`
   show all projects

2. `python gui.py -c config_gui.ini --sql_host`
   show all hosts and the corresponding projects

3.
   ```
   python gui.py -c config_gui.ini --sql_error
   --start_date 01.01.2005 --end_date 01.03.2005
   --ip 134.225.4.18
   ```
   show all errors of SRB system with the IP 134.225.4.18 between 01.01.2005 and 01.03.2005

4.
   ```
   python gui.py -c config_gui.ini --sql_error
   --start_date 01.01.2005  --project mySRBproject
   ```
   show all errors between 01.01.2005 and now for the project “mySRBproject”

5.
   ```
   python gui.py -c config_gui.ini --sql_error
   --start_date 22.10.2005 --end_date 22.10.2005
   --start_time 12:00:00 --end_time 18:00:00
   --ip 127.0.0.1 --file test.txt
   ```
   show all errors on the 22.10.2005 between 12 h and 18 h on localhost and save output in file “test.txt”

6.
   ```
   python gui.py -c config_gui.ini --sql_error_freq
   --error -1023 --ip 127.0.0.1 -g
   ```
   show error frequency for the error -1023 from host 127.0.0.1 and display diagram (graph)

C.4 Remote Controller

The Remote Controller can be used to influence the parsing behaviour of the server. The application is a console application only and requires Python 2.2.3 or higher, and M2Crypto 0.13. The parameter to control the server are presented in Table C.6.

TABLE C.6: Remote Controller Parameter

Parameter	Explanation
	general parameters
-h or –help	print help
-c or –config	defines configuration file
-g or –graph	show output additionally as a diagram
–nocolor	no coloured console output
	server commands
–rpc_status	show actual setting of RPC (disabled/enabled)
–disable_rpc	disable RPC
–enable_rpc	enable RPC
–shutdown	shutdown server
–change_interval <int>	change parsing interval of server
–keyword_status	show actual setting of keywords
–add_keyword <string>	add keyword to keyword list
–delete_keyword <string>	delete keyword in keyword list
–ignore_error_status	show actual setting of "ignore_error"
–add_ignore_error <int>	add error, which the parser should ignore
–delete_ignore_error <int>	delete error, which the parser is ignoring
	additional parameters
–ip <ip>	host IP (*e.g.* 127.0.0.1)
–port <int>	port, where the server is listening

The server commands can only be used once at a time. The additional commands can be combined. The necessary configuration file contains only name and path for the certificate file and certificate authority file.

Before the application is executed with

```
python admin_server -c configuratin_file.ini [ parameter ]
```

the configuration file should be adjusted. After the application was started the required action is executed and the server's answer is displayed. The application terminates after the task is processed.

C.4.1 Examples

For a better understand some simple examples are given:

1. `python admin_server -c config_admin_server.ini --rpc_status --ip 134.225.4.18 --port 6000`
   request RPC status from server with IP 134.225.4.18 which listens on port 6000
2. `python admin_server -c config_admin_server.ini --change_interval 60 --ip 134.225.4.18 --port 6000`
   change parsing interval time to 60 minutes at server with IP 134.225.4.18
3. `python admin_server -c config_admin_server.ini --delete_keyword status --ip 134.225.4.18 --port 6000`
   delete keyword "status" in keyword file at server with IP 134.225.4.18
4. `python admin_server -c config_admin_server.ini --add_ignore_error 5,6 --ip 134.225.4.18 --port 6000`
   add new error numbers 5 and 6 which are to be ignored at server with IP 134.225.4.18
5. `python admin_server -c config_admin_server.ini --shutdown --ip 134.225.4.18 --port 6000`
   shutdown server with IP 134.225.4.18

C.5 GZ Parser

The GZ Parser application is used to process older SRB log files which are only available as compressed files. The application requires Python 2.2.3 or higher. The parameters are displayed in Table C.7.

TABLE C.7: GZ Parser Parameters

Parameter	Explanation
-h or –help	print help
-c or –config	defines configuration file
-v or –verbose	activates printing of messages [debug option]

Before the application is started with

```
python gz_parser.py -c configuration_file.ini [-v | -h]
```

the configuration file has to be adjusted. Keywords can be defined in the keyword file. Table C.8 explains the items in the configuration file.

TABLE C.8: Configuration File GZ Parser

Parameter	Explanation
	Section Files
keyword	name of keyword file
	Section Path
path_srb_gz	path of SRB server gz log files
path_xml_file	location (path) of the XML file within the server environment
path_keyword	path of keyword file
	Section Misc
ignore_error	some error numbers might not be interesting, so the errors should be ignored, *e.g.* 0, 3, 5 (comma separated list)

It should be paid attention to the fact that the XML file path is very important. Only if the application places the XML file within the server XML directory, the client is able to fetch the files.

Appendix D

Source Code

D.1 Server

D.1.1 Module start_server.py

LISTING D.1: Module start_server.py

```
#!/usr/bin/env python
'''
This module is the log file parser server start script.

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

import server_classes
import sys, os, time, getopt
import socket
import fcntl
import struct
from utils_server import LoadConfig, check_ip, usage_exit

class WorkingServer:
    '''
    This is the main class for the server.
    '''

    def __init__(self, config_data, config_dir, verb):
        '''
        Constructor
        '''
        self._verbose = verb
        config = config_data
        self._workingpath = os.getcwd()

        self._server_certificate = config.get("files.server_certificate")
        self._server_certificate_path = config.get("path.path_server_certificate")
        self._server_certificate_path = self._server_certificate_path.rstrip("/")
        if(self._server_certificate_path == '' or self._server_certificate_path == None):
            self._server_certificate_path = self._workingpath
        else:
            self._server_certificate = self._server_certificate.strip()
            if (-1 != self._server_certificate_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._server_certificate_path = self._workingpath+"/"+self._server_certificate_path

        self._server_ca = config.get("files.server_ca")
        self._server_ca_path = config.get("path.path_server_ca")
        self._server_ca_path = self._server_ca_path.rstrip("/")
        if(self._server_ca_path == '' or self._server_ca_path == None):
            self._server_ca_path = self._workingpath
        else:
            self._server_ca = self._server_ca.strip()
            if (-1 != self._server_ca_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._server_ca_path = self._workingpath+"/"+self._server_ca_path

        self._srb_log = config.get("files.srb_log")
        self._srb_log_path = config.get("path.path_srb_log")
        self._srb_log_path = self._srb_log_path.rstrip("/")
        if(self._srb_log_path == '' or self._srb_log_path == None):
            self._srb_log_path = self._workingpath
        else:
            self._srb_log = self._srb_log.strip()
            if (-1 != self._srb_log_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._srb_log_path = self._workingpath+"/"+self._srb_log_path

        self._gz_path = config.get("path.path_gz")
        self._gz_path = self._gz_path.rstrip("/")
        if(self._gz_path == '' or self._gz_path == None):
            self._gz_path = self._workingpath
        else:
            if (-1 != self._gz_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._gz_path = self._workingpath+"/"+self._gz_path

        self._keyword_name = config.get("files.keyword")
        self._keyword_path = config.get("path.path_keyword")
        self._keyword_path = self._keyword_path.rstrip("/")
        if(self._keyword_path == '' or self._keyword_path == None):
            self._keyword_path = self._workingpath
        else:
            self._keyword_name = self._keyword_name.strip()
            if (-1 != self._keyword_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._keyword_path = self._workingpath+"/"+self._keyword_path

        self._xml_file = "client_log.xml"
        self._xml_file_path = os.getcwd()+"/xml_client"

        try:
            self._interval = int(config.get("misc.minute"))
        except ValueError:
            print "Please check the configuration in the config file (section: misc, item: minute). It should have the following pattern:\nminute = <int>"
            os._exit(-1)
        try:
            self._port = int(config.get("misc.port"))
        except ValueError:
            print "Please check the configuration in the config file (section: misc, item: port). It should have the following pattern:\nport = <int>"
            os._exit(-1)
        if (self._port < 1024 or self._port > 50001):
            print "A server port is out of range. \nPlease check the configuration file and make sure the server port lies between 1025 (inclusive) and 50000 (inclusive)!\n\n"
            os._exit(-1)

         #check if the configuration if correct
        if(0 == os.path.exists(self._server_certificate_path+"/"+self._server_certificate)):
            print "Could not locate server certifiate under %s !\nMaybe change configuration file and try again!\n\n" % self._server_certificate_path
            os._exit(-1)

        if(0 == os.path.exists(self._server_ca_path+"/"+self._server_ca)):
            print "Could not locate server ca certificate under %s !\nMaybe change configuration file and try again!\n\n" % self._server_ca_path
            os._exit(-1)

        if(0 == os.access((self._srb_log_path+"/"+self._srb_log), 4)):     # 4 R_OK
```

```
            print "Could not access SRB server log file under %s !\nMaybe change configuration file and try again!\n\n" % self._srb_log_path
            os._exit(-1)

        if(0 == os.path.exists(self._xml_file_path)):
            print "Creating path \"%s\"\n\n" % self._xml_file_path
            os.mkdir(self._xml_file_path)

        self._share = server_classes.Mutex()

        self._keyword = server_classes.get_keywords(self._keyword_path+"/"+self._keyword_name)

        error = config.get("misc.ignore_error")

        self._ip = config.get("misc.interface")

        error = error.strip()
        error = error.strip(",")
        self._ignore_error = error.split(",")
        for i in range(len(self._ignore_error)):
            if self._ignore_error[i] != '':
                try:
                    self._ignore_error[i] = int(self._ignore_error[i].strip())
                except ValueError:
                    print "Please check the \"ignore_error\" list in the config file (section: misc). It should have the following pattern (comma separeted list of integer):\nignore_error = <int>,<int> "
                    os._exit(-1)
            else:
                del self._ignore_error[i]

        self._configfile = config_dir

        self._rpc = server_classes.RPC(self._verbose, self._share, self._configfile, self._interval, self._keyword_path, self._keyword_name, self._xml_file_path)

    def establish_connection(self):
        '''
        establish a working connection using MySSLServer
        '''
        cert = self._server_certificate_path+"/"+self._server_certificate
        ca = self._server_ca_path+"/"+self._server_ca
        self._serverobject = server_classes.My_SSL_Server(cert, ca, self._verbose)
        ip = self.get_ip_address(self._ip)
        if (0 == check_ip(ip)):
            self._serv = self._serverobject.start_server(ip, self._port)
        else:
            print "Could not start the server. (IP: \"%s\")" % ip
            os._exit(-1)

        # start thread for parsing log file
        workerthread = server_classes.MyParserThread(self._share, self._interval, self._gz_path, self._srb_log_path, self._srb_log, self._keyword, self._ignore_error, self._xml_file_path, self._xml_file, self._configfile, (self._keyword_path+"/"+self._keyword_name), self._verbose)
        workerthread.setName("parser")
        workerthread.start()
        print "Started!\n\n"

    def get_ip_address(self, network_interface):
        '''
        Uses the Linux SIOCGIFADDR ioctl to find the IP address associated with a network interface, given the name of that interface, e.g. "eth0".

        source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/439094

        modified by a.weise (December 2005)
        '''
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        try:
            ip = socket.inet_ntoa(fcntl.ioctl(
                s.fileno(),
                0x8915,  # SIOCGIFADDR
                struct.pack('256s', network_interface[:15])
            )[20:24])
            return ip
        except IOError, e:
            return e

    def register_functions(self):
        '''
        register all the rpc - functions
        '''
        self._serv.register_function(self._rpc.rpc_stop_server, 'stop_server')
        self._serv.register_function(self._rpc.rpc_status, 'rpc_status')
        self._serv.register_function(self._rpc.rpc_disable_rpc_calls, 'disable_rpc_calls')
        self._serv.register_function(self._rpc.rpc_enable_rpc_calls, 'enable_rpc_calls')
        self._serv.register_function(self._rpc.rpc_get_my_xml_file, 'get_my_xml_file')
        self._serv.register_function(self._rpc.rpc_get_file_list, 'get_file_list')
        self._serv.register_function(self._rpc.rpc_update_configuration, 'rpc_update_configuration')
        self._serv.register_function(self._rpc.rpc_update_keyword_file, 'rpc_update_keyword_file')
        self._serv.register_function(self._rpc.rpc_check_availabitity, 'rpc_check_availabitity')
        self._serv.register_function(self._rpc.rpc_interval_status, 'rpc_interval_status')

    def run_server(self):
        '''
        handle all client requests
        '''
        try:
            #serv.serve_forever()
            if self._verbose == 1:
                print "\nSimple SSL XML RPC Server is running ....\n"
            while (1):
                if self._verbose == 1:
                    print "%s -> waiting for request ...." % time.ctime()
                self._serv.handle_request(self._serv)
        except KeyboardInterrupt:
            # if the server is not running as a daemon shutdown with Ctrl+c is possible
            if self._verbose == 1:
                sys.stdout.write("\n\nShutdown !!!\n\n")

            command = "./stop_server"
            os.system(command)


################################################################################


def daemonize(verbose, stdout = '/dev/null', stderr = None, stdin = '/dev/null', pidfile = None, startmsg = 'Server daemon started with pid %s'):

    '''
    This function creates a daemon by forking the current process. The parameters stdin, stdout, and stderr are file names which substitute the standard err-, in-, out- output. This parameters are optional and point normally to /dev/null. Note that stderr is opened unbuffered, so if it shares a file with stdout then interleaved output may not appear in the order that you expect.

    source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/66012
    modified by a.weise November 2005
    '''

    # first fork => fork creates first child-process
    try:
        pid = os.fork()
        if (pid > 0):
            sys.exit(0) # close first parent-process
    except OSError, e:
        sys.stderr.write("fork #1 failed: (%d) %s\n" % (e.errno, e.strerror))
        sys.exit(1)

    os.umask(0)
    os.setsid()

    # second fork
    try:
        pid = os.fork()
        if (pid > 0):
            sys.exit(0) # close second parent-process
    except OSError, e:

        sys.stderr.write("fork #2 failed: (%d) %s\n" % (e.errno, e.strerror))
        sys.exit(1)

    # open standard in and out and print standard message
    if (not stderr):# if not stderr given => take stdout-path
        stderr = stdout

    if verbose == 1:
        si = file(stdin, 'r')
        so = file(stdout, 'w+') # w -> overwrite old log content
        se = file(stderr, 'w+', 0)
        pid = str(os.getpid())
        sys.stderr.write("\n%s\n" % startmsg % pid)
        sys.stderr.flush()
        if pidfile:
            file(pidfile,'w+').write("%s\n" % pid)

        # redirect standard in and out to files
        os.dup2(si.fileno(), sys.stdin.fileno())
        os.dup2(so.fileno(), sys.stdout.fileno())
        os.dup2(se.fileno(), sys.stderr.fileno())

###################################################################

def start():

    '''
    START THE APPLICATION
    '''
    configfile = ""
    verbose = 0
    daemon = 0

    try:
        opts, args = getopt.getopt(sys.argv[1:], 'c:vhd', ['config=', 'verbose', 'help', 'daemon'])
        for opt, value in opts:
            if opt in ('-h','--help'):
                msg = "\n\t\t-----------   Help   ----------\n\n\n"\
                      "-c or --config\t->  defines config file, if no config file given, default values are used\n"\
                      "-v or --verbose\t->  activates printing of messages [debug option]\n"\
                      "-d or --daemon\t->  daemonize the server\n"\
                      "-h or --help\t->  print this help\n\n"
                usage_exit(sys.argv[0], msg)
            if opt in ('-c','--config'):
                value = value.replace("=", "")
                configfile = os.getcwd()+"/"+value
            if opt in ('-v','--verbose'):
                verbose = 1
            if opt in ('-d', '--daemon'):
                daemon = 1
    except getopt.error, e:
        usage_exit(sys.argv[0], e)

    # load config file or default values
    if (configfile != ""):
        # check if file exists
        if(1 == os.path.exists(configfile)):
            config = LoadConfig(configfile)
        else:
            # if file NOT exists terminate program
            print "Sorry, the given file does NOT exist !\nPlease try again!\n\n"
            os._exit(-1)
    else:
        msg = "\nNo configuration file spezified !\n"
        usage_exit(sys.argv[0], msg)

    print "\n\n--------------------- SRB LOG FILE PARSER [ SERVER ] ------------------ \n\n"
    print "Starting ..."

    worker = WorkingServer(config, configfile, verbose)

    if daemon == 1:
        if verbose == 1:
            #if verbose then write messages in log file
            daemonize(verbose, stdout = 'daemonize.log')
        else:
            # quit mode
            daemonize(verbose)
    else:
        pass

    worker.establish_connection()
    worker.register_functions()
    worker.run_server()


if __name__ == '__main__':

    start()
```

D.1.2 Module server_classes.py

LISTING D.2: Module `server_classes.py`

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

'''
This module contains all necessary classes and functions for the gz_parser.py and srb log file parser -> start_server.py .

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

# ssl connection
from SimpleXMLRPCServer import SimpleXMLRPCServer, SimpleXMLRPCRequestHandler
from M2Crypto import SSL

# misc
import os, time, stat

# regular expressions
import re

# utilities
from utils_server import delete_file, list_to_string, get_keywords, LoadConfig, find

# threads
import thread
import threading

import socket

################# CLASS SimpleSSLXMLRPCServer #################

class SimpleSSLXMLRPCServer(SSL.SSLServer, SimpleXMLRPCServer):
    '''
    This class is derived from SSL.SSLServer and SimpleXMLRPCServer.
    '''
    def __init__(self, ssl_context, address, verbose, handler=SimpleXMLRPCRequestHandler):
        '''
        Constructor overwrites the init function of the SimpleXMLRPCServer and replace it with the secure SSLServer.
        '''
        SSL.SSLServer.__init__(self, address, handler, ssl_context)
        self.funcs = {}
        self.logRequests = 0
        self.instance = None
        self._verbose = verbose

    def handle_request(self, serv):
        '''
        Handle one request by passing it on the a thread.
        '''
        try:
            request, client_address = self.get_request()
            if self._verbose == 1:
                print "%s -> request accepted from %s....." % (time.ctime(), client_address[0])

        except socket.error:
            return
        if self.verify_request(request, client_address):
            thd = MyClientThread(request, client_address, serv)
            thd.start()

################## CLASS My_SSL_Server ##################

class My_SSL_Server:
    '''
    provide functions for the server class
    '''

    def __init__(self, server_cert, ca_cert, verbose):
        '''
        Constructor
        '''
        self._server_certificate = server_cert
        self._ca_certificate = ca_cert
        self._verbose = verbose

    def start_server(self, address, port, xmlrpcserver=SimpleSSLXMLRPCServer):
        '''
        Start the actual server using SSL:

        sslv23 -> compatibility mode, can handle any of the three SSL/TLS protocol versions
        server.pem -> server certificate including server RSA private key
        ca.pem -> root certificate
        SSL.verify_none -> no request that the client has to send his certificate as well
        '''
        # create SSL context
        ctx = self.init_context('sslv3', self._verbose, self._server_certificate, self._ca_certificate, SSL.verify_none)
        # create server object
        server = xmlrpcserver(ctx, (address, port), self._verbose)
        # return server object
        return server

    def init_context(self, protocol, verbose, certfile, cafile, verify, verify_depth=10):
        '''
        This function is used to generate the SSL context:
```

```
        - verify_depth -> chain depth
        '''
        ctx = SSL.Context(protocol)   # create context object
        ctx.load_cert_chain(certfile)   # load server certificate chain
        ctx.load_verify_locations(cafile)
        ctx.set_client_CA_list_from_file(cafile)
        ctx.set_verify(verify, verify_depth)  # verfiy options
        ctx.set_session_id_ctx('server')        # set session id
        #if verbose == 1:
        #    ctx.set_info_callback()    # show handshake information --- debug
        return ctx

################## CLASS MyClientThread ##################

class MyClientThread(threading.Thread):
    '''
    This class presents a client, which connects to the server.
    '''

    def __init__(self, request, address, serv_object):
        '''
        Constructor
        '''
        self._request = request
        self._client = address
        self._serv = serv_object
        threading.Thread.__init__(self)

    def run(self):
        '''
        This function overrides the standard run method.
        '''
        try:
            self._serv.process_request(self._request, self._client)
            self._serv.close_request(self._request)
        except:
            self._serv.handle_error(self._request, self._client)
            self._serv.close_request(self._request)

################## CLASS LogFileParser ##################

class LogFileParser:
    '''
    This class provides all the neccessarey tools to parse and work up to logfile of the SRB-System.
    '''

    def __init__(self, logfilepath, keywords, ignore_error, xml_file_path, xml_file_name, verbose):
        '''
        Constructor
        '''
        self._verbose = verbose
        self._ignore_error = ignore_error
        self._keywords = keywords  # keywords
        self._logfile_path = logfilepath
        self._client_log_file = "%s/%s" % (xml_file_path, xml_file_name)  # name and path of client log file
        self._client_log_file_fd = -1      # client log file - file descriptor
        self._first_line = range(15)   # save first 15 lines of log file
        self._last_byte_number = 0     # save last byte number which was parsed
        self._line_number     = 1         #save last line number which was parsed

        if(0 == os.path.exists(self._logfile_path)):
            print "Could not locate log file path under %s !\nMaybe change configuration file and try again!\n\n" % self._logfile_path
            os._exit(-1)

    def _fetch_first_lines(self, file_name):
        '''
        This function returns the first 15 lines from current logfile without saving them anywhere.
        '''
        try:
            log_file_fd = open(file_name, "r")

            listline = range(15)
            log_file_fd.seek(0) #set cursor on first position
            for i in range(15):
                listline[i] = log_file_fd.readline()
            log_file_fd.close()
            return listline
        except IOError:
            print "Could not open file -> ", file_name

    def set_first_lines(self, file_name):
        '''
        This function saves the first 15 lines of the log file into the member variable.
        '''
        try:
            log_file_fd = open(file_name, 'r')
            log_file_fd.seek(0) #set cursor on first position
            for i in range(15):
                self._first_line[i] = log_file_fd.readline()
            log_file_fd.close()
        except IOError:
            if self._verbose == 1:
                print "%s -> Could not open file -> \"%s\"" % (time.ctime(), file_name)

    def get_first_lines(self):
        '''
        This function returns the member variable _first_lines.
        '''
        return self._first_line

    def test_first_lines(self, file_name):
        '''
        This function compares the first 15 lines of a log file and return 0 if they are the same, otherwise -1.
        '''
        listline = self._fetch_first_lines(file_name)
        z = 0
        while(z<15):
            if(listline == self._first_line):
                z += 1
            else:
                return -1

        return 0

    def find_size_last_message_tag(self, fd, msg_tag):
        '''
        This function find out, how many bytes the last message tag needs. This function was necessary, because this new xml messages have to be added into the client xml file. Since the creating of this file is not very straightforward using known techniques like sax or dom, the last tag gets deleted, the new messages added and the last tag writen again.

        fd = file descriptor of the file
        msg_tag = message tag to search for
        '''
        #set cursor back
        z = -1
        fd.seek(0, 2)

        while(1):
            fd.seek(z, 2)
            tag = fd.read()
            if (-1 != tag.rfind(msg_tag)):
                return z
            z -= 1
            if -15 == z:
                # message tag was not part of the file
                return 0

    def analyse_log_file(self, parser_file_name, file_time=None):
        '''
        takes the templog file and goes through each lines and searches for keywords, if keywords are found, the line and the two lines before and after are dumped into a xml file, which the client can collect. This function uses the system function write to create the xml file. The dom function were to ineffectiv and sax unflexible.

        parser_file_name = file name of the file, which needs to be parsed
        '''
        # determine year of paser file, needed for extract time, since log file content does not provide a year
        if file_time != None:
            tupel = time.gmtime(file_time)
        else:
            status = os.stat(parser_file_name)
            file_time = status[8]
            tupel = time.gmtime(file_time)

        pf_year = time.strftime("%Y", tupel)

        byte_count = 0
        interrupt = 0
        z = 0

        try:
            self._client_log_file_fd = file(self._client_log_file, 'r+')

        except IOError:
            # create new client log file
            self._client_log_file_fd = open(self._client_log_file, 'w')

            xml_header = "<?xml version=\"1.0\" encoding=\"utf-8\" standalone=\"yes\"?>\n"
            self._client_log_file_fd.writelines(xml_header)
            self._client_log_file_fd.writelines("<message>\n")
            self._client_log_file_fd.writelines("</message>\n")
            self._client_log_file_fd.close()
            self._client_log_file_fd = file(self._client_log_file, 'r+')


        # set cursor in file
        x = self.find_size_last_message_tag(self._client_log_file_fd, "</message>")

        self._client_log_file_fd.seek(x, 2)
        shorten = self._client_log_file_fd.tell()
        # delete last </message>
        self._client_log_file_fd.truncate(shorten)
        self._client_log_file_fd.seek(0, 2)

        #open log file
        log_file = parser_file_name

        try:
            log_file_fd = open(log_file, "r")
            log_file_fd.seek(self._last_byte_number)

        except IOError:
            if self._verbose == 1:
                print "%s -> could not open srb log file -> %s" % (time.ctime(), log_file)
            return -1

        if self._verbose == 1:
            print "%s -> start parsing " % (time.ctime())
        starttime = time.time()
        while(interrupt == 0):
            # read line
            content = log_file_fd.readline()
            if(content == ''):
                interrupt = 1
                break

            if 0 == len(self._keywords):
                check = 0
            else:
                check = self._test_keywords(self._keywords, len(self._keywords)-1, content)

            if(0 == check):
                # extract error number
                error_number = self._extract_error_number(content)

                if (None == error_number):
                    error_number = "-"
                    temp = ""
                else:
                    temp = int(error_number)

                if ( None == find(temp, self._ignore_error)):

                    line_number_string = "%d" % self._line_number
                    # delete whitespace
                    content_ = content.rstrip()

                    date_time = self._extract_time(content, pf_year)

                    if (date_time == -1):
                        date_time = ["", ""]
                        # save current byte count
                        byte_count = log_file_fd.tell()
                        back = -1
                        read = 0
                        while(1):
                            try:
                                # find time pattern by going back character by character
                                log_file_fd.seek(back, 1)
                                read = back*(-1)
                                tag = log_file_fd.read(read)
                                if (None != re.search('^NOTICE: *[A-Z][a-z]{2} +[0-9]{1,2} +[0-9]{2}:[0-9]{2}:[0-9]{2}:', tag)):
                                    date_time = self._extract_time(tag, pf_year)
                                    break
                                back = back-1
                            except IOError:
                                # no time available
                                date_time[0] = time.strftime("%Y-%m-%d", tupel)
                                date_time[1] = time.strftime("%H:%M:%S", tupel)

                                break

                        #restore byte count
                        log_file_fd.seek(byte_count)

                    z += 1 # entry counter

                    if -1 == self.start_entry('entry'):
                        interrupt = 1
                        break
                    if -1 == self.write_entry('date', date_time[0]):
                        interrupt = 1
                        break
                    if -1 == self.write_entry('time', date_time[1]):
                        interrupt = 1
                        break
                    if -1 == self.write_entry('error_number', error_number):
                        interrupt = 1
                        break
                    if -1 == self.write_entry('error_string', content_):
                        interrupt = 1
                        break
                    if -1 == self.write_entry('linenumber', line_number_string):
                        interrupt = 1
                        break
                    if -1 == self.end_entry('entry'):
                        interrupt = 1
                        break

            self._line_number += 1
        if self._verbose == 1:
            print "%s -> end parsing " % (time.ctime())
        endtime = time.time()
        if self._verbose == 1:
            print "%s -> parsing time: %s" % (time.ctime(), (endtime-starttime))
        self.end_entry('message')
        if self._verbose == 1:
            print "%s -> %d errors found\n" % (time.ctime(), z) #--- debug ---
        # save last byte number
        try:
            self._client_log_file_fd.close()
        except IOError, e:
            if self._verbose == 1:
                print "%s -> Problem closing XML file: \"%s\" !" % (time.ctime(), e)
        self._last_byte_number = log_file_fd.tell()
```

```

    def start_entry(self, name):
        '''
        This function inserts a start tag into the XML file. (name = tag name)
        '''
        start_tag = "<%s>\n" % name
        try:
            self._client_log_file_fd.write(start_tag)
            return 0
        except IOError, e:
            if self._verbose == 1:
                print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(), e)
            return -1

    def write_entry(self, tagname, content):
        '''
        This function inserts an entry into the xml file.

        tagname = tag name
        content = message between start and end tag
        '''

        if len(content) < 50000000:
            #find all not allowed character  old: [\x09\x0a\x0d\x20-\xd7]*
            bad_character = re.sub('[\x09\x0a\x0d\x20-\x25\x27-\xd7]*', "", content)
            # replace each not allowed character with "?"
            for i in range(len(bad_character)):
                if bad_character[i] == '\x00':
                    # delete NUL character
                    content = content.replace(bad_character[i], '')
                else:
                    content = content.replace(bad_character[i], "?")

            entry = "<%s>%s</%s>\n" % (tagname, content, tagname)
            try:
                self._client_log_file_fd.write(entry)
                return 0
            except IOError, e:
                if self._verbose == 1:
                    print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(), e)
                return -1

        else:
            entry = "<%s>LOGFILE ENTRY TO LONG !!!</%s>\n" % (tagname, tagname)
            try:
                self._client_log_file_fd.write(entry)
                return 0
            except IOError, e:
                if self._verbose == 1:
                    print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(), e)
                return -1

    def end_entry(self, name):
        '''
        This function inserts an end tag into the XML file. (name = tag name)
        '''
        endtag = "</%s>\n" % name
        try:
            self._client_log_file_fd.write(endtag)
            return 0
        except IOError, e:
            if self._verbose == 1:
                print "%s -> Problem writing XML file: \"%s\" !" % (time.ctime(), e)
            return -1

    def reset(self):
        '''
        This function resets member variable, in case of a new log file.
        '''
        self._line_number = 1
        self._last_byte_number = 0

    def _test_keywords(self, keywordlist, amount_of_keywords, teststring):
        '''
        This is a recursive function, which tests if a list of keywords is part of a string (AND relation). If all keywords found 0 is returned, otherwise -1

        keywordlist = list of all keywords
        amount_of_keywords = number of keywords in list
        teststring = string, which needs to be investigated

        return -1 if line is not interesting
        return 0 if line is taken
        '''
        if (amount_of_keywords == 0):
            #last keyword check   -1 != content.rfind(”NOTICE”)
            if( 2 == len(keywordlist[amount_of_keywords])):
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return 0
                else:
                    if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
                        # check for NO keyword
                        temp = keywordlist[amount_of_keywords][1].strip("!")
                        if ( -1 == teststring.rfind(temp)):
                            # go on to next keyword
                            return 0
                        else:
                            return -1
                    else:
                        # there is no ”!”
                        if ( -1 != teststring.rfind(keywordlist[amount_of_keywords][1])):
                            # string is there, go on to next keyword
                            return 0
                        else:
                            return -1
            else:
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return 0
                else:
                    return -1
        else:
            if( 2 == len(keywordlist[amount_of_keywords])):
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                else:
                    if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
                        # check for NO keyword
                        temp = keywordlist[amount_of_keywords][1].strip("!")
                        if ( -1 == teststring.rfind(temp)):
                            # go on to next keyword
                            return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                        else:
                            return -1
                    else:
                        # there is no "!"
                        if ( -1 != teststring.rfind(keywordlist[amount_of_keywords][1])):
                            # string is there, go on to next keyword
                            return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                        else:
                            return -1
            else:
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                else:
                    return -1

    def update_keywords(self, keys):
        '''
        This function updates the member variable keywords.

        keys = new keyword list
        '''
        self._keywords = keys

    def update_ignore_error(self, error):
        '''
        This function updates the ignore_error list.

        error = new ignore list
        '''
        self._ignore_error = error

    def _extract_time(self, time_string, year):
        '''
        This function takes a line from the logfile and extracts the time from there.
        '''
        if ( None == re.search('^NOTICE: *[A-Z][a-z]{2} +[0-9]{1,2} +[0-9]{2}:[0-9]{2}:[0-9]{2}:', time_string)):
            return -1
        else:
            listus = time_string.split(":")
            zeit = year+listus[1]+":"+listus[2]+":"+listus[3]
            time_tupel = time.strptime(zeit, "%Y %b %d %X")
            date_time = range(2)
            date_time[0] = time.strftime("%Y-%m-%d", time_tupel)
            date_time[1] = time.strftime("%H:%M:%S", time_tupel)
            return date_time

    def _extract_error_number(self, text):
        '''
        Thhis function takes a line from the logfile and extract the error number.
        '''
        match = re.search('(status|errno) *= *-*\d{1,10}', text)
        if(None != match):
            #if match then give me number
            listus = match.string[match.start():match.end()]
            listus = re.findall('-*\d{1,10}', listus)
            if(1 == len(listus)):
                return listus[0]
            else:
                return None
        else:
            return None

################# CLASS MyParserThread #################

class MyParserThread(threading.Thread):
    '''
    This class is used to create a thread, which is doing all the necessery work in the background.
    '''
    def __init__(self, shared, timus, gz_path, logfilepath, logfilename, keyword, ignore_error, xml_file_path, xml_file_name, configfile, keywordfile, verbose):
        '''
        Constructor
        '''
        self._keywordfile = keywordfile
        temp = os.stat(self._keywordfile)
        self._keywordfile_time = temp[8]
        self._configfile = configfile
        temp = os.stat(self._configfile)
        self._configfile_time = temp[8]
        self._verbose = verbose
        self._id = thread.get_ident()
        self._shared_obj = shared
        self._interval = timus
        self._stop = 0
        threading.Thread.__init__(self)
        self._parser = LogFileParser (logfilepath, keyword, ignore_error, xml_file_path, xml_file_name, self._verbose)
        self._logfilepath = logfilepath
        self._log_file_name = logfilepath+"/"+logfilename
        self._gz_diect = gz_path
        self._list = [] #save *.gz files

    def run(self):
        '''
        This function overwrites the standard run method.
        '''
        block_counter = 0
        while(1):
            # acquire lock
            if self._stop == 1:
                print "%s -> working thread stopped !!!" % time.ctime()
                os._exit(0)
            # check if config file has changed
            if (self._configfile != ""):
                temp = os.stat(self._configfile)
                if self._configfile_time != temp[8]:
                    #if time has changed save new time
                    if self._verbose == 1:
                        print "%s -> config file has changed, reading new values ..." % time.ctime()
                    self._configfile_time = temp[8]
                    self._refresh_configuration()

            # check if keyword file has changed
            if (self._keywordfile != ''):
                temp = os.stat(self._keywordfile)
                if self._keywordfile_time != temp[8]:
                    #if time has changed
                    if self._verbose == 1:
                        print "%s -> keyword file has changed, reading new values ..." % time.ctime()
                    self._keywordfile_time = temp[8]
                    self._refresh_keywords()

            if(-1 == self._shared_obj.set_variable_parsing(1, self)):
                # client is busy
                block_counter += 1
                time.sleep(5*block_counter)
                if self._verbose == 1:
                    print "%s -> client busy" % time.ctime()
                if block_counter > 5:
                    if self._verbose == 1:
                        print "%s -> client needs a long time, miss this parsing period" % time.ctime()
                    block_counter = 0
                    time.sleep(self._interval*60)
            else:
                # no client busy
                try:

                    # check if the first lines the same
                    if(0 == self._parser.test_first_lines(self._log_file_name)):
                        #if the first 15 lines still the same
                        if self._verbose == 1:
                            print "%s -> no log file rotation" % time.ctime()
                        self._parser.analyse_log_file(self._log_file_name)
                    else:
                        if self._verbose == 1:
                            print "%s -> new log file" % time.ctime()
                        # create gz file list
                        # empty list for the gz_files
                        self._list = []

                        os.path.walk(self._gz_diect, self.parse_directory, self._list)
                        if (0 < len(self._list)):
                            self.gunzip(self._list[0][1])
                            d = os.getcwd()
                            try:
                                os.chdir(self._gz_diect)
                                self.gunzip(self._list[0][1])
                                self._parser.analyse_log_file(self._gz_diect+"/temp_srbLog")
                                delete_file("temp_srbLog", self._verbose)
                                os.chdir(d)
                            except:
                                if self._verbose == 1:
                                    print "%s -> could not find directory \"%s\"" % (time.ctime(), self._gz_diect)
                        else:
                            pass

                        self._parser.reset()
                        self._parser.set_first_lines(self._log_file_name)
                        self._parser.analyse_log_file(self._log_file_name)

                finally:

                    # release lock
                    self._shared_obj.set_variable_parsing(0, self)

            time.sleep(self._interval*60)#

            if self._verbose == 1:
                print "\n%s -> -------- parse ------" % time.ctime() #--- debug ---

    def _refresh_keywords(self):
        '''
        This function gets keywords from the keyword file !
        '''
        if(1 == os.path.exists(self._keywordfile)):
            keyword = get_keywords(self._keywordfile)
            self._parser.update_keywords(keyword)
            return 0
        else:
            # if file NOT use old configuration
            if self._verbose == 1:
                print "%s -> Sorry, the keyword file does NOT exist !\nUsing old configuration!\n\n" % time.ctime()
            return -1

    def _refresh_configuration(self):
        '''
        This function gets the needed information from the configfile!
        '''
        if(1 == os.path.exists(self._configfile)):
            config = LoadConfig(self._configfile)
            error = config.get("misc.ignore_error")
            error = error.strip()
            error = error.strip(",")
            ignore_error = error.split(",")
            for i in range(len(ignore_error)):
                # check if there is an entry at all
                if ignore_error[i] != '':
                    try:
                        ignore_error[i] = int(ignore_error[i].strip())
                    except ValueError:
                        if self._verbose == 1:
                            print "%s -> \"ignore_error\" in the config file has NO valid values, use old configuration" % time.ctime()
                            return -1
                else:
                    del ignore_error[i]

            self._parser.update_ignore_error(ignore_error)

            return 0
        else:
            # if file NOT exists terminate program
            if self._verbose == 1:
                print "%s -> Sorry, the given config file does NOT exist !\nUsing old configuration!\n\n" % time.ctime()
            return -1

    def stop_thread(self):
        """
        Stop the thread
        """
        self._stop = 1

    def parse_directory(self, arg, dirname, fnames):
        '''
        This function "walks" through a given directory and considers all srbLOG*.gz files. The name and last modified time are saved in a list (2 dimensional array). The function should be used with os.path.walk(path, function_name, arg)!
        '''
        d = os.getcwd()
        # change into log file directory
        try:
            os.chdir(dirname)
        except:
            if self._verbose == 1:
                print "%s -> could not find directory \"%s\"" % (time.ctime(), dirname)
            return -1
        # for each file
        for f in fnames:
            # check if file and if file is a log file e.g. srbLog.20051003.gz
            if (not os.path.isfile(f)) or (None == re.search('^srbLog[_0-9.-]*.gz', f)):
                continue
            # get last modified time
            date = os.stat(f)[stat.ST_MTIME]
            # create tupel
            tupel = (date, f)
            # save last modified time and filename into am arrray (list)
            self._list.append(tupel)
        # change back into the working directory
        os.chdir(d)
        # sort list ascending (aufsteigend)
        self._list.sort()
        # reverse list order, sorted descending (absteigend), the greater the time number the younger the file
        self._list.reverse()
        return 0

    def gunzip(self, filus, name_temp_file="temp_srbLog"):
        '''
        This function unzips a *.gz file using the system tool gunzip. Make sure when calling the function the file exists in this directory. The function creates a temporary file and leave the orignal *.gz file untouched!
        '''
        if (not os.path.isfile(filus)):
            return -1
        else:
            command = "gunzip -c %s > %s" % (filus, name_temp_file)
            try:
                os.system(command)
                return 0
            except:
                return -1

#########################  C L A S S   M U T E X   ######################

class Mutex:
    '''
    This class makes sure that server and client are not accessing the same file at the same time.
    '''
    # lock
    _locked = threading.Lock()


    def __init__(self):
        '''
        Constructor
        '''
        self.parsing = 0
        self._parsing_thread_id = 0
        self.client = 0


    def set_variable_parsing(self, value, the_thread):
        '''
        set variable parsing
        '''
        Mutex._locked.acquire()  # lock
        self._parsing_thread_id = the_thread

        if self.client == 0:
            #set variable
            self.parsing = value
            Mutex._locked.release()
            time.sleep(1)
            if value == 0:
                # reset parsing thread identity
                self._parsing_thread_id = 0
            return 0
        else:
            Mutex._locked.release()   # release lock
            time.sleep(1)
            return -1

    def set_variable_client(self, value):
        '''
        set variable client
        '''
        Mutex._locked.acquire()  # lock
        # if client is not fetching the file
        if self.parsing == 0:
            #set variable
            self.client = value
            #print "client variable gesetzt"
            Mutex._locked.release()   # release lock
            time.sleep(1)
            return 0
        else:
            if (0 != self._parsing_thread_id):
                if (1 != self._parsing_thread_id.isAlive()):
                    # if parsing thread dead, reset semphore
                    self.parsing = 0
                    self._parsing_thread_id = 0
            Mutex._locked.release()   # release lock
            time.sleep(1)
            return -1

########################   C L A S S    R P C    ######################

class RPC:
    '''
    This class contains the RPC functions.
    '''
    def __init__(self, verbose, semaphore, config_file, interval, keyword_path,
        keyword_name, xml_file_path):
        '''
        constructor
        '''
        self._verbose = verbose
        self._client = True
        self._share = semaphore
        self._config_file = config_file
        self._interval = interval
        self._keyword_path = keyword_path
        self._keyword_name = keyword_name
        self._xml_file_path = xml_file_path

        self._list = [] #for walking through the directory

    def rpc_stop_server(self):
        '''
        This function stops the server!
        '''
        command = "./stop_server"

        answer = os.system(command)
        print answer
        return answer

    def rpc_disable_rpc_calls(self):
        '''
        This function disables rpc.
        '''
        self._client = False
        if self._verbose == 1:
            print "%s -> RPC through \"admin tool\" disabled!" % time.ctime()
        return "RPC disabled"

    def rpc_enable_rpc_calls(self):
        '''
        This function enables rpc.
        '''
        self._client = True
        if self._verbose == 1:
            print "%s -> RPC through \"admin tool\" enabled!" % time.ctime()
        return "RPC enabled"

    def rpc_status(self):
        '''
        This function return the current status of the self._client variable.
        '''
        if self._client == True:
            return "RPC enabled"
        else:
            return "RPC disabled"

    def rpc_interval_status(self):
        '''
        This function returns the current parsing interval time.
        '''
        if self._client == True:
            return self._interval
        else:
            return -2

    def rpc_get_my_xml_file(self, filename):
        '''
        This functions gets the xml file from the server !
        '''
        if (self._client == True):
            if (0 == self._share.set_variable_client(1)):
                # check if file is available
                try:
                    filus = self._xml_file_path+"/"+filename
                    client_xml_fd = open(filus, 'r')
                    file_content = client_xml_fd.read()
                    client_xml_fd.close()
                except IOError:
                    self._share.set_variable_client(0)
                    return "no file"
                #delete xml file
                if (0 == delete_file((self._xml_file_path+"/"+filename), self._verbose)):
                    self._share.set_variable_client(0) # reset variable
                    return file_content
                else:
                    if self._verbose == 1:
                        print "problems deleting file"
                    self._share.set_variable_client(0)
                    return -1
            else:
                return -3   # server is busy parsing
        else:
            return -2 # rpc disalbed

    def rpc_check_availabitity(self):
        '''
        This function check if the server is still in the parsing process.
        '''
        if (self._client == True):
            if (0 == self._share.set_variable_client(1)):
                return 0
            else:
                return -3
        else:
            return -2

    def rpc_get_file_list(self):
        '''
        This function walks through the *.xml directory and finds all files, which need to be fetched form the client.
        '''
        if (self._client == True):
            self._list = [] # empty list
            try:
                os.path.walk(self._xml_file_path, self._parse_directory, self._list)
                if (0 < len(self._list)):
                    return self._list
                else:
                    return 0
            except:
                return -1
        else:
            return -2 # rpc disalbed

    def rpc_update_configuration(self, section, key, value, action):
        '''
        This functions adds or deletes values in the config.ini.
        action:
        0 = delete
        1 = add
        2 = exchange
        4 = info
        '''
        print "section: ", section
        print "key: ", key
        print "value: ", value
        print "action: ", action
        if(self._client == True):
            try:
                config_fd = file(self._config_file, 'r+')
            except IOError, e:
                return "Problem -> %s" % e

            byte_count = 0

            while(1):
                byte_count = config_fd.tell()
                line = config_fd.readline()
                if line == '':
                    if self._verbose == 1:
                        print "%s -> Section: \"%s\" and key: \"%s\" do not exist in config file!" % (time.ctime(), section, key)
                    config_fd.close()
                    return "Section: \"%s\" and key: \"%s\" do not exist in config file!" % (section, key)
                if (-1 != line.find(section)):
                    while(1):
                        byte_count = config_fd.tell()
                        line = config_fd.readline()
                        if line == '':
                            if self._verbose == 1:
                                print "%s -> Key: \"%s\" do not exist under section \"%s\" in config file!" % (time.ctime(), key, section)
                            config_fd.close()
                            return "Key \"%s\" do not exist under section \"%s\" in config file!" % (key, section)
                        if (-1 != line.find(key) and -1 != line.find("=") and -1 == line.find("#", 0, 1)):
                            if action == 0:
                                if self._verbose == 1:
                                    print "%s -> Delete \"%s:%s\" value \"%s\"" % (time.ctime(), section, key, value)
                                listus = line.split("=")
                                listus[1] = listus[1].strip()
                                listus[1] = listus[1].strip(",")
                                listus = listus[1].split(",")
                                for i in range(len(listus)):
```

```
                listus[i] = listus[i].strip()
            new_content = ''
            for i in range(len(listus)):
                if int(listus[i]) != value:
                    new_content = new_content+"%s, " % listus[i]

            new_content = new_content.strip()
            new_content = new_content.strip(",")
            new_content = "%s = %s\n" % (key, new_content)
            rest = config_fd.read()
            config_fd.truncate(byte_count)
            config_fd.seek(byte_count)
            config_fd.writelines(new_content)
            config_fd.write(rest)
            config_fd.close()
            return "Changes applied: %s" % new_content
        elif action == 1:
            if self._verbose == 1:
                print "%s -> Add \"%s:%s\" value \"%s\"" % (time.ctime(), section, key, value)
            listus = line.split("=")
            listus[1] = listus[1].strip()
            listus[1] = listus[1].strip(",")
            listus = listus[1].split(",")
            finish = 0
            while (finish == 0):
                if len(listus) == 0:
                    finish = 1
                for i in range(len(listus)):
                    finish = 1 # break the while loop
                    print listus[i]
                    listus[i] = listus[i].strip()

                    try:
                        print "test auf int"
                        temp_value = int(listus[i])
                        if value == temp_value:
                            config_fd.close()
                            return "Value %d already exists!" % value
                    except ValueError, e:
                        finish = 0 # activate while loop
                        print e, listus[i]
                        del listus[i]
                        break


            new_content = ''
            for i in range(len(listus)):
                new_content = new_content+"%s, " % listus[i]
            new_content = "%s = %s%s\n" % (key, new_content, value)
```

```
                                rest = config_fd.read()
                                config_fd.truncate(byte_count)
                                config_fd.seek(byte_count)
                                config_fd.writelines(new_content)
                                config_fd.write(rest)
                                config_fd.close()
                                return "Changes applied: %s" % new_content
                            elif action == 2:
                                # exchange
                                if self._verbose == 1:
                                    print "%s -> Change \"%s:%s\" to value \"%s\"" %
                                         (time.ctime(), section, key, value)
                                rest = config_fd.read()
                                config_fd.truncate(byte_count)
                                config_fd.seek(byte_count)
                                new_content = "%s = %s\n" % (key, value)
                                config_fd.writelines(new_content)
                                config_fd.write(rest)
                                config_fd.close()
                                if section == 'misc' and key == 'minute':
                                    self._interval = int(value)
                                return 0
                            elif action == 4:
                                config_fd.close()
                                return line
                            else:
                                config_fd.close()
                                return -1
        else:
            return "RPC disabled" # rpc disalbed

    def rpc_update_keyword_file(self, keyword, action):
        '''
        This function updates the keyword file.

        action:
        0 = delete
        1 = add
        2 = info
        '''
        if(self._client == True):
            byte_count = 0
            file_size = 0
            comments = ''

            filus = self._keyword_path+"/"+self._keyword_name

            try:
                key_fd = file(filus, 'r+')
            except IOError, e:
                if self._verbose == 1:
```

```
                print "%s -> Problem open keyword file -> %s !" % (time.ctime(), e)
            return "Problem -> %s" % e

        key_fd.seek(0, 2) # set cursor to end of file
        file_size = key_fd.tell()
        key_fd.seek(0) # set cursor to begining of file

        while(1):
            # get comments
            byte_count = key_fd.tell()
            line = key_fd.readline()
            if byte_count >= file_size:
                break
            if (-1 != line.find("#", 0, 1)):
                comments += line

        key_fd.close()
        keyword_list = get_keywords(filus)
        keyword = keyword.split(":")

        if action == 0:
            # test if keyword is already there
            for i in range(len(keyword_list)):
                if 1 == len(keyword) and 1 == len(keyword_list[i]):
                    if keyword[0] == keyword_list[i][0]:
                        del keyword_list[i]
                        try:
                            key_fd = file(filus, 'w+')
                            print key_fd
                            key_string = list_to_string(keyword_list)
                            key_fd.write(comments)
                            key_fd.write(key_string)
                            key_fd.close()
                        except IOError, e:
                            if self._verbose == 1:
                                print "%s -> Problem open keyword file -> %s !" % (time.ctime(), e)
                            return "Problem -> %s" % e
                        return "keyword %s from keyword list deleted" % keyword
                elif 2 == len(keyword) and 2 == len(keyword_list[i]):
                    if keyword[0] == keyword_list[i][0] and keyword[1] == keyword_list[i][1]:
                        del keyword_list[i]
                        try:
                            key_fd = file(filus, 'w+')
                            key_string = list_to_string(keyword_list)
                            key_fd.write(comments)
                            key_fd.write(key_string)
                            key_fd.close()
                        except IOError, e:
                            if self._verbose == 1:
                                print "%s -> Problem open keyword file -> %s !" % (time.ctime(), e)
                            return "Problem -> %s" % e
                        return "keyword %s from keyword list deleted" % keyword

            return "keyword %s was not part of keyword list" % keyword

        if action == 1:
            # test if keyword is already there
            for i in range(len(keyword_list)):
                if 1 == len(keyword) and 1 == len(keyword_list[i]):
                    if keyword[0] == keyword_list[i][0]:
                        return "keyword %s already in keyword list" % keyword
                elif 2 == len(keyword) and 2 == len(keyword_list[i]):
                    if keyword[0] == keyword_list[i][0] and keyword[1] == keyword_list[i][1]:
                        return "keywords %s already in keyword list" % keyword

            keyword_list.append(keyword)
            try:
                key_fd = file(filus, 'w+')
            except IOError, e:
                if self._verbose == 1:
                    print "%s -> Problem open keyword file -> %s !" % (time.ctime(), e)
                return "Problem -> %s" % e

            key_string = list_to_string(keyword_list)
            key_fd.write(comments)
            key_fd.write(key_string)
            key_fd.close()

            return key_string

        if action == 2:
            return list_to_string(keyword_list)

    else:
        return "RPC disabled"

def _parse_directory(self, arg, dirname, fnames):
    '''
    This function "walks" through a given directory and looks for the client_log.xml file. The name and last modified time are saved in a list (2 dimensional array). The function should be used with os.path.walk(path, function_name, arg)!

    dirname = directory which need to be pared
    fnames = files within dirname
    '''
    d = os.getcwd()
    # change into log file directory
    try:
        os.chdir(dirname)
    except:
        if self._verbose == 1:
            print "could not find directory \"%s\"" % dirname
        return -1
    # for each file
    for f in fnames:
        # check if file and if file is a log file e.g. client_log.xml
        if (not os.path.isfile(f)) or (None == re.search('client_log.xml', f)):
            continue
        else:
        # save filename into an arrray (list)
            self._list.append(f)
    # change back into the working directory
    os.chdir(d)
```

D.1.3 Module utils_server.py

LISTING D.3: Module utils_server.py

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

'''
This module provides basic utilities for the modules server_classes.py and start_server.py.

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

import ConfigParser, string
import time, os

def LoadConfig(file_name, config={}):
    """
    returns a dictionary with key's of the form
    <section>.<option> and the values

    source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/65334
    """
    config = config.copy()
    cp = ConfigParser.ConfigParser()
    cp.read(file_name)
    for sec in cp.sections():
        name = string.lower(sec)
        for opt in cp.options(sec):
            config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
    return config

def check_ip(ip):
    '''
    This function checks if a given IP is valid.
    '''
    try:
        ip = ip.split(".")
    except AttributeError:
        return -1

    for i in range(len(ip)):
        check =  ip[i].find("0", 0, 1)
        if -1 != check and 1 < len(ip[i]):
            return -1
        try:
            ip[i] = int(ip[i])
        except ValueError:
            return -1
        if ip[i] >= 0 and ip[i] <= 255:
            pass
        else:
            return -1

    return 0

def get_keywords(filus):
    '''
    This function extracts keywords from a give file!
    '''
    keys = []

    try:
        file_fd = file(filus, 'r')
    except IOError, e:
        print "Problem with keyword file -> ",  e
        return -1

    content = file_fd.readlines()# save file content as list (1 line == 1 entry)

    file_fd.close()

    content = remove_item(content, "#")
    content = remove_item(content, "\n")

    for i in range(len(content)):
        content[i] = content[i].strip()
        content[i] = content[i].rstrip(",")
        content[i] = content[i].split(",")
        for a in range(len(content[i])):
            keys.append(content[i][a])

    for i in range(len(keys)):
        keys[i] = keys[i].strip() # remove whitespace
        keys[i] = keys[i].split(":")

    return keys

def remove_item(listus, item):
    '''
    This function removes an item for a list (2 dimentional) as a rekursive function.
    '''

    while(1):

        for i in range(len(listus)):
            if -1 != listus[i].find(item, 0, 1):
                del listus[i]
                remove_item(listus, item)
                break
        else:
            break

    return listus

def list_to_string(listus):
    '''
    This function converts the keyword list (2 dimensional array) to a keyword string (keywords comma separated), so the string is writable into the keyword file.
    '''
    str_listus = ''

    for i in range(len(listus)):
        if 1 == len(listus[i]):
            str_listus += listus[i][0]+", "
        elif 2 == len(listus[i]):
            str_listus += listus[i][0]+":"+listus[i][1]+", "

    str_listus = str_listus.strip()
    str_listus = str_listus.strip(",")

    return str_listus

def delete_file(file_name, verbose):
    '''
    This function deletes a file.
    '''
    try:
        os.remove(file_name)
        return 0
    except:
        if verbose == 1:
            print "%s -> could not delete -> \"%s\"" % (time.ctime(), file_name)
        return -1
```

```

def usage_exit(progname, msg=None):
    '''
    This function diplays the usage of this program and terminates the program!
    '''
    if msg:
        print msg
        print
    print "usage: python %s [ -h|--help -c|--config -v|--verbose -d|--daemon] \n\n" % progname
    os._exit(-1)

def find(search, listus):
    '''
    This function finds an item within a list (1 dimensional).
    '''
    for i in range(len(listus)):
        if listus[i] == search:
            return listus[i]
    return None
```

D.1.4 Script `stop_server.sh`

Listing D.4: Script `stop_server.sh`

```
#!/bin/sh
#
# Script to shutdown server
#
# Reading University
# MSc in Network Centered Computing
# a.weise - a.weise@reading.ac.uk - December 2005
#
echo "stopping server ...."
name=start_server.py

# Find all servers
server_pid=`ps -elf | egrep $name | egrep -v grep | awk '{ print $4 }'`

if [ "$server_pid" = "" ]
then
  echo No server is running !
else
  /bin/kill -15 $server_pid
  server_pid=`ps -elf | egrep $name | egrep -v grep | awk '{ print $4 }'`
  if [ "$server_pid" = "" ]
  then
    echo server stopped
  else
```

```
    /bin/kill -9 $server_pid
    echo server killed
  fi
fi
```

D.2 Client

D.2.1 Module start_client.py

LISTING D.5: Module start_client.py

```
#!/usr/bin/env python
'''
This module is the log file parser client start file.

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

import client_classes, os, sys, time
import  getopt, smtplib, socket
from utils_client import LoadConfig, check_ip, usage_exit, get_password

class MyClient:
    '''
    main class for the client application
    '''
    def __init__(self, config_data, verb, smtp_pa):
        '''
        Constructor
        '''
        self._verbose = verb
        config = config_data
        self._workingpath = os.getcwd()

        #-------------------- put together path and file --------------------------------------------
        self._database_name = config.get("database.name")
        self._database_path = config.get("database.path")
        self._database_path = self._database_path.rstrip("/")
        if(config.get("database.path") == '' or config.get("database.path") == None):
            # field is empty
            self._database_path = self._workingpath

        else:
            self._database_name = self._database_name.strip()
            if (-1 != self._database_path.find("/", 0, 1)):
                # first character "/"
                print "/ an erster stelle"
                #pass
            else:
                self._database_path = self._workingpath+"/"+self._database_path

        self._error_description_name = config.get("files.error_description")
        self._error_description_path = config.get("path.path_error_description")
        self._error_description_path = self._error_description_path.rstrip("/")
        if(config.get("path.path_error_description") == '' or config.get("path.path_error_description") == None):
            self._error_description_path = self._workingpath
        else:
            self._error_description_name = self._error_description_name.strip()
            if (-1 != self._error_description_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._error_description_path = self._workingpath+"/"+self._error_description_path

        self._client_certificate = config.get("files.client_certificate")
        self._client_certificate_path = config.get("path.path_client_certificate")
        self._client_certificate_path = self._client_certificate_path.rstrip("/")
        if(config.get("path.path_client_certificate") == '' or config.get("path.path_client_certificate") == None):
            self._client_certificate_path = self._workingpath
        else:
            self._client_certificate = self._client_certificate.strip()
            if (-1 != self._client_certificate_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._client_certificate_path = self._workingpath+"/"+self._client_certificate_path

        self._client_ca = config.get("files.client_ca")
        self._client_ca_path = config.get("path.path_client_ca")
        self._client_ca_path = self._client_ca_path.rstrip("/")
        if(config.get("path.path_client_ca") == '' or config.get("path.path_client_ca") == None):
            self._client_ca_path = self._workingpath
        else:
            self._client_ca = self._client_ca.strip()
            if (-1 != self._client_ca_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._client_ca_path = self._workingpath+"/"+self._client_ca_path

        # check if the configuration if correct
        if(0 == os.path.exists(self._client_certificate_path+"/"+self._client_certificate)):
            print "Could not locate client certifiate under %s !\nMaybe change configuration file and try again!\n\n" % self._client_certificate_path
            os._exit(-1)

        if(0 == os.path.exists(self._client_ca_path+"/"+self._client_ca)):
            print "Could not locate client ca certificate under %s !\nMaybe change configuration file and try again!\n\n" % self._client_ca_path
            os._exit(-1)

        if(0 == os.path.exists(self._error_description_path+"/"+self._error_description_name)):
            print "Could not locate error description file under %s !\nMaybe change configuration file and try again!\n\n" % self._error_description_path
            os._exit(-1)

        self._project = config.get("project.name")
        self._interval = int(config.get("misc.minute"))

        #------------------ create server list ---------------------------
        servers = config.get("server.serverlist")
        # split where commas
        servers_split = servers.split(",")
        # create dictionary
        self._serverlist = {}    # dictionary for serverlist:port
        for i in range(len(servers_split)):
            # remove whitespace
            servers_split[i] = servers_split[i].strip()
            temp_list = servers_split[i].split(":")
            # remove whitespace
            if len(temp_list) != 2:
                print "The IP configuration \"%s\" seems not correct. \nPlease check the configuration file!\n\n" % temp_list[0]
                os._exit(-1)
            temp_list[0] = temp_list[0].strip()
            temp_list[1] = temp_list[1].strip()
            # check if IP is valid
            if( -1 == check_ip(temp_list[0])):
                print "The IP \"%s\" seems not correct. \nPlease check the configuration file!\n\n" % temp_list[0]
                os._exit(-1)
            try:
                temp_list[1] = int(temp_list[1])
            except ValueError:
                print "The port \"%s\" is not valid.\nPlease check the configuration file!\n\n" % temp_list[1]
                os._exit(-1)
            if (temp_list[1] < 1024 or temp_list[1] > 50001):
                print "A server port is out of range. \nPlease check the configuration file and make sure the server port lies between
```

```
                    1025 (inclusive) and 50000 (inclusive)!\n\n"
                os._exit(-1)
            self._serverlist[temp_list[0]] = temp_list[1]

        self._share = client_classes.Mutex()

        # mail issues
        self._smtp_server = config.get("mail.smtp_server")
        self._smtp_pass = smtp_pa
        self._smtp_from = config.get("mail.from")
        self._smtp_user = config.get("mail.user")

        if (None != self._smtp_pass):
            # test if smtp server and login is possible
            try:
                if self._verbose == 1:
                    print "Test SMTP connection to server \"%s\"...." % self.
                        _smtp_server
                server = smtplib.SMTP(self._smtp_server)
                print "server object: ", server
                if self._verbose == 1:   # --- debug ---
                    server.set_debuglevel(1) # ---- debug ----
                server.login(self._smtp_user, self._smtp_pass)
                server.quit()
                if self._verbose == 1:
                    print "SMTP connection successfully tested"
            except smtplib.SMTPAuthenticationError, e:
                print "Problem with SMTP server authentication -> \"%s\" !" % e
                print "\n"
                os._exit(-1)
            except socket.error, e:
                print "Problem with SMTP server -> \"%s\" !" % e
                print "\n"
                os._exit(-1)

        self._mail_address = []    # mail address list
        z = 1
        while(1):
            temp = "mail_to.address_%d" % z
            testus = config.get(temp)
            if testus == None:
                break
            self._mail_address.append(testus)
            z += 1

        keywordfiles = []

        z = 1
        while(1):
            temp = "mail_to.path_ignore_error_%d" % z
            path = config.get(temp)
            if(path == '' or path == None):
                path = self._workingpath
            else:
                path = path.rstrip("/")
                if (-1 != path.find("/", 0, 1)):
                    # first character "/"
                    pass
                else:
                    path = self._workingpath+"/"+path

            temp = "mail_to.file_ignore_error_%d" % z
            filus = config.get(temp)
            if filus == None:
                break
            filus = filus.strip()

            keywordfilus = path+"/"+filus

            if(0 == os.access(keywordfilus, 4)):     # 4 -> R_OK -> read only
                print "Could not access keyword file under %s !\nMaybe change
                    configuration file and try again!\n\n" % keywordfilus
                os._exit(-1)

            keywordfiles.append(keywordfilus)
            z += 1

        self._mail_ignore_error = range(len(keywordfiles))
        for i in range(len(keywordfiles)):
            self._mail_ignore_error[i] = self._get_keywords(keywordfiles[i])

    def _get_keywords(self, filus):
        '''
        This function extracts keyword from a give file!
        '''
        keys = []

        try:
            file_fd = file(filus, 'r')
        except IOError, e:
            print "Problem with keyword file -> ", e
            return -1

        content = file_fd.readlines()# save file contetn as list (1 line == 1 entry)

        file_fd.close()

        content = self._remove_item(content, "#")
        content = self._remove_item(content, "\n")

        for i in range(len(content)):
            content[i] = content[i].strip()
            content[i] = content[i].rstrip(",")
            content[i] = content[i].split(",")
            for a in range(len(content[i])):
                keys.append(content[i][a])

        for i in range(len(keys)):
            keys[i] = keys[i].strip() # remove whitespace
            keys[i] = keys[i].split(":")

        return keys

    def _remove_item(self, listus, item):
        '''
        This function removes an item for a list as a rekursive function.
        '''
        while(1):

            for i in range(len(listus)):
                if -1 != listus[i].find(item, 0, 1):
                    del listus[i]
                    self._remove_item(listus, item)
                    break
            else:
                break

        return listus

    def initialise_database(self):
        '''
        This function is initalising the database, creates it, when it's not there! It creates finally the database access cursor for further work with the database.
        '''
        self._db = client_classes.MyDatabase(self._error_description_name, self._error_description_path, self._database_name, self._database_path, self._serverlist.items(), self._project, self._verbose)

    def get_serverlist(self):
        '''
        This function returns the server list.
        '''
        return self._serverlist

    def fetch_error_messages(self):
        '''
        This function starts the worker thread, who initalises the regular fetching of the error messages.
        '''
        self._workerthread = client_classes.WorkerThread(self._share, self._db, self._interval, self._serverlist.items(), self._client_certificate, self._client_certificate_path, self._client_ca, self._client_ca_path, self._verbose, self._mail_address, self._mail_ignore_error, self._smtp_server, self._smtp_pass, self._smtp_from, self._smtp_user)
        self._workerthread.setName("workerthreadDaemon")
        self._workerthread.start()

        if self._verbose == 1:
            print "%s -> Manager thread started !" % ( time.ctime() ) #--- debug ----


#########################################################

def daemonize(verbose, stdout = '/dev/null', stderr = None, stdin = '/dev/null',
    pidfile = None, startmsg = 'Client daemon started with pid %s'):

    '''
    This function creates a daemon by forking the current process. The parameters
        stdin, stdout, and stderr are file names which substitute the standard err-,
        in-, out- output. This parameters are optional and point normally to /dev/
        null. Note that stderr is opened unbuffered, so if it shares a file with
        stdout then interleaved output may not appear in the order that you expect.

    source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/66012
    modified by a.weise November 2005
    '''

    # first fork => fork creates first child-process
    try:
        pid = os.fork()
        if (pid > 0):
            sys.exit(0) # close first parent-process

    except OSError, e:
        sys.stderr.write("fork #1 failed: (%d) %s\n" % (e.errno, e.strerror))
        sys.exit(1)

    os.umask(0)
    os.setsid()

    # second fork
    try:
        pid = os.fork()
        if (pid > 0):
            sys.exit(0) # close second parent-process

    except OSError, e:

        sys.stderr.write("fork #2 failed: (%d) %s\n" % (e.errno, e.strerror))
        sys.exit(1)

    # open standard in and out and print standard message
    if (not stderr):# if not stderr given => take stdout-path
        stderr = stdout


    if verbose == 1:
        si = file(stdin, 'r')
        so = file(stdout, 'w+') # w -> overwrite old log content
        se = file(stderr, 'w+', 0)
        pid = str(os.getpid())
        sys.stderr.write("\n%s\n" % startmsg % pid)
        sys.stderr.flush()
        if pidfile:
            file(pidfile,'w+').write("%s\n" % pid)

        # redirect standard in and out to files
        os.dup2(si.fileno(), sys.stdin.fileno())
        os.dup2(so.fileno(), sys.stdout.fileno())
        os.dup2(se.fileno(), sys.stderr.fileno())



##################################################################

def start():

    '''
    Start the application.
    '''
    configfile = ""
    verbose = 0
    smtp_pass = None
    daemon = 0

    try:
        opts, args = getopt.getopt(sys.argv[1:], 'c:vhpd', ['config=', 'verbose', 'help', 'smtp_password', '--daemon'])
        for opt, value in opts:
            if opt in ('-h','--help'):
                msg = "\n-----------   Help   ----------\n\n\n"\
                      "-c or --config\t\t->  defines config file, if no config file given, default values are used\n"\
                      "-p or --smtp_password\t->  activates mail notification sending\n"\
                      "-v or --verbose\t\t->  activates printing of messages [debug option]\n"\
                      "-d or --daemon\t\t->  daemonize the client\n"\
                      "-h or --help\t\t->  print this help\n\n"
                usage_exit(sys.argv[0], msg)
            if opt in ('-c','--config'):
                value = value.replace("=", "")
                configfile = os.getcwd()+"/"+value
            if opt in ('-v','--verbose'):
                verbose = 1
            if opt in ('-p', '--smtp_password'):
                smtp_pass = get_password("Please enter SMTP password: ")
            if opt in ('-d', '--daemon'):
                daemon = 1
    except getopt.error, e:
        usage_exit(sys.argv[0], e)

    # load config file or default values
    if (configfile != ""):
        # check if file exists
        if(1 == os.path.exists(configfile)):
            config = LoadConfig(configfile)
        else:
            # if file NOT exists terminate program
            print "Sorry, a given file does NOT exist !\nPlease try again!\n\n"
            os._exit(-1)
    else:
        msg = "\nNo config file spezified !\n"
        usage_exit(sys.argv[0], msg)

    print "\n\n--------------------- SRB LOG FILE PARSER [ CLIENT ] ------------------ \n\n"
    print "Starting ..."


    worker = MyClient(config, verbose, smtp_pass)
    worker.initialise_database()

    if daemon == 1:
        if verbose == 1:
            daemonize(verbose, stdout = 'daemonize.log')
        else:
            daemonize(verbose)
    else:
        pass

    print "%s -> Start manager thread ..." % (time.ctime())
    worker.fetch_error_messages()

if __name__ == '__main__':

    start()
```

D.2.2 Module client_classes.py

Listing D.6: Module client_classes.py

```
#!/usr/bin/env python
'''
This module contains all imports, defines and basic classes for start_client.py.

Reading University
MSc in Network Centered Computing
```

```
a.weise - a.weise@reading.ac.uk - December 2005
'''
# misc
import os, sys, signal, re, copy
import string, time

# database
import sqlite

#mail
import smtplib, socket

# xml parsing
from xml.sax import make_parser
from xml.sax.handler import ContentHandler, feature_namespaces
import xml.sax

# connection issues
from M2Crypto.m2xmlrpclib import Server, SSL_Transport
from M2Crypto import SSL

# threads
import threading, thread

################# CLASS MyContentHandler #################

class MyContentHandler(ContentHandler):
    '''
    This class is derived from _xmlplus.sax.handler and provides individual functions for parsing the xml file.
    '''

    def __init__(self, db_object, ip, ignore_error, mail_obj, verbose):
        '''
        Constructor
        '''
        self._verbose = verbose
        self._my_mail_ignore_error = ignore_error
        self._mail_obj = mail_obj
        self._ip = ip
        self._db = db_object
        self._db_access = self._db.get_access_cursor()
        self._searchTerm = ""
        self._date = ""
        self._date_flag = 0
        self._time = ""
        self._time_flag = 0
        self._error_number = 0
        self._error_number_flag = 0
        self._error_string = ""
        self._error_string_flag = 0
        self._linenumber = 0
        self._linenumber_flag = 0

    def set_ip(self, ip):
        '''
        The function sets the member variable _ip.
        '''
        self._ip = ip

    def startElement(self, tag, attr):
        '''
        The function overwrites the startElement function.
        '''
        self._searchTerm = tag

    def characters (self, tag_text):
        '''
        This function overwrites the character function to extract the tag content.
        '''
        if (self._searchTerm == "date"):
            self._date = tag_text
            self._date_flag = 1
        elif (self._searchTerm == "time"):
            self._time = tag_text
            self._time_flag = 1
        elif (self._searchTerm == "error_number"):
            self._error_number = tag_text
            self._error_number_flag = 1
        elif (self._searchTerm == "error_string"):
            self._error_string = tag_text
            self._error_string_flag = 1
        elif (self._searchTerm == "linenumber"):
            self._linenumber = tag_text
            self._linenumber_flag = 1

    def endElement(self, tag):
        '''
        This function overwrites endElement function.
        '''
        if (self._searchTerm == "date"):
            pass
        elif (self._searchTerm == "time"):
            pass
        elif (self._searchTerm == "error_number"):
            pass
        elif (self._searchTerm == "error_string"):
            self._error_string = self._error_string.replace("\"", "")
        elif (self._searchTerm == "linenumber"):
            pass
        self._searchTerm = "" #reset variable

        if(self._date_flag == 1 and self._time_flag == 1 and self._error_number_flag
            == 1 and self._error_string_flag == 1 and self._linenumber_flag == 1):
```

```
            # save in database
            success = self._insert()
            if success == -1:
                # raise exception to exit
                print " raise exception"
                assert success == 0

            # add mail content
            if (0 != len(self._mail_obj)):
                for i in range(len(self._mail_obj)):
                    check = self._test_keywords(self._my_mail_ignore_error[i], len(self._my_mail_ignore_error[i])-1, self._error_string)
                    if (0 == check):
                        # print " add mail content"
                        cont = "\n-------------------------------------"\
                               "\ndate:\t\t\t"+self._date+ \
                               "\ntime:\t\t\t"+self._time+ \
                               "\nerror message:\t\t"+self._error_string+ \
                               "\nline number:\t\t"+self._linenumber
                        # add mail content
                        self._mail_obj[i][0].add(cont)
                        # modify error counter
                        self._mail_obj[i][0].count()
                        # set first date
                        temp = "%s (%s)" % (self._date, self._time)
                        if self._mail_obj[i][0].get_first_date() == '':
                            self._mail_obj[i][0].set_first_date(temp)
                        # set last date
                        self._mail_obj[i][0].set_last_date(temp)

            # reset variables
            self._reset()

    def _test_keywords(self, keywordlist, amount_of_keywords, teststring):
        '''
        This is a recursive function, which tests if a list of keywords is part of a string (AND relation). If all keywords found 0 is returned, otherwise -1

        keywordlist = list of all keywords
        amount_of_keywords = number of keywords in list
        teststring = string, which needs to be investigated

        return -1 if line is not interesting
        return 0 if line is taken
        '''
        if (amount_of_keywords == 0):
            #last keyword check   -1 != content.rfind("NOTICE")
            if( 2 == len(keywordlist[amount_of_keywords])):
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return 0
                else:
                    if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
                        # check for NO keyword
                        temp = keywordlist[amount_of_keywords][1].strip("!")
                        if ( -1 == teststring.rfind(temp)):
                            # go on to next keyword
                            return 0
                        else:
                            return -1
                    else:
                        # there is no "!"
                        if ( -1 != teststring.rfind(keywordlist[amount_of_keywords][1])):
                            # string is there, go on to next keyword
                            return 0
                        else:
                            return -1
            else:
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return 0
                else:
                    return -1
        else:
            if( 2 == len(keywordlist[amount_of_keywords])):
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                else:
                    if( -1 == keywordlist[amount_of_keywords][1].rfind("!")):
                        # check for NO keyword
                        temp = keywordlist[amount_of_keywords][1].strip("!")
                        if ( -1 == teststring.rfind(temp)):
                            # go on to next keyword
                            return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                        else:
                            return -1
                    else:
                        # there is no "!"
                        if ( -1 != teststring.rfind(keywordlist[amount_of_keywords][1])):
                            # string is there, go on to next keyword
                            return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                        else:
                            return -1
            else:
                if (-1 == teststring.rfind(keywordlist[amount_of_keywords][0])):
                    # not in string go to next keyword
                    return self._test_keywords(keywordlist, amount_of_keywords-1, teststring)
                else:
                    return -1

    def _reset(self):
        '''
        This function resets member variables.
        '''
        self._date = ""
        self._date_flag = 0
        self._time = ""
        self._time_flag = 0
        self._error_number = 0
        self._error_number_flag = 0
        self._error_string = ""
        self._error_string_flag = 0
        self._linenumber = 0
        self._linenumber_flag = 0

    def _insert(self):
        '''
        This function inserts the data from the xml file into the database.
        '''
        # get first error number id !!!!!!!!!!!!!!!!!!!!
        if ('-' == self._error_number):
            #if no error number
            self._error_number = 999999

        sql = ' SELECT * FROM error WHERE e_number = "%s" ' % self._error_number
        success, self._db_access = self._db.execute_sql(1200, self._db_access, sql)
        if success == -1:
            return -1
        data = self._db_access.fetchall()
        if ( 0 == len(data)):
            # error number not in database -> insert new error number into database
            sql = ' INSERT INTO error (e_number, e_name, e_description) VALUES ("%s", "not specified", "") ' % self._error_number
            data = self._db_access.execute(sql)
            sql = ' SELECT * FROM error WHERE e_number = "%s" ' % self._error_number
            success, self._db.execute_sql(1200, self._db_access, sql)
            if success == -1:
                return -1
            data = self._db_access.fetchall()

        error_id = data[0]["e_id"]

        # check if dataset already there
        sql = 'SELECT * FROM messages WHERE error_e_id = "%s"'% error_id+ \
            ' AND m_date = "%s" '% self._date + \
            ' AND m_time = "%s" '% self._time + \
      ' AND m_error_string = "%s"' %( self._error_string)

        success, self._db.execute_sql(1200, self._db_access, sql)
        if success == -1:
            return -1
        data = self._db_access.fetchall()
        if (0 == len(data)):
            # if dataset is not in database insert it
            # 2. get host id
            sql = 'SELECT * FROM host WHERE h_ip_address = "%s";' % self._ip
            success, self._db.execute_sql(1200, self._db_access, sql)
            if success == -1:
                return -1
            data = self._db_access.fetchall()
            if (1 == len(data)):
                ip = data[0]["h_id"]
            else:
                ip = data[0]["h_id"]
            # insert data in database
            sql = 'INSERT INTO messages (host_h_id, error_e_id, m_date, m_time, m_error_string, m_line_number) VALUES (%s, %s, "%s", "%s", "%s", %s);' % (ip, error_id, self._date, self._time, self._error_string, self._linenumber)
            success, self._db.execute_sql(1200, self._db_access, sql)
            if success == -1:
                return -1
        else:
            pass

        return 0

################## CLASS Mail ##################

class Mail:
    '''
    This class deals with the mail issues.
    '''

    def __init__(self, mail_address, smtp_server, smtp_pass, smtp_from, user, verbose):
        '''
        Constructor
        '''
        self._verbose = verbose
        self._mail_address = mail_address
        self._smtp_server = smtp_server
        self._smtp_pass = smtp_pass
        self._smtp_from = smtp_from
        self._smtp_user = user
        self._mail_name = "temp_email_unknown.txt"

        self._error_count = 0
        self._first_date = ''
        self._last_date = ''
        self._first_date_flag = 0



    def create_content(self, name):
        '''
        This function creates a temorary file, where the mail content gets saved
            temporarly.
        '''
        try:
            file_fd = open(name, 'w')
            self._mail_name = name
            file_fd.close()
            return 0
        except IOError, e:
            if self._verbose == 1:
                print "%s -> Problem creating email content -> " % (time.ctime(), e)
            return -1

    def add(self, content):
        '''
        This function adds to the mail content.
        '''
        try:
            file_fd = file(self._mail_name, 'r+')
            file_fd.seek(0, 2) # cursor to end of file
            file_fd.writelines(content)
            file_fd.close()
            return 0
        except IOError, e:
            if self._verbose == 1:
                print "%s -> Problem adding email content -> " % (time.ctime(), e)
            return -1

    def count(self):
        '''
        This function counts all inserted error within the mail by incrementing the
            member variable self._error_count.
        '''
        self._error_count += 1

    def set_first_date(self, value):
        '''
        This function modifies the memeber variable self._first_date.
        '''
        self._first_date = value

    def get_first_date(self):
        '''
        This function returns the content of the memeber variable self._first_date.
        '''
        return self._first_date

    def set_last_date(self, value):
        '''
        This function modifies the memeber variable self._last_date
        '''
        self._last_date = value

    def send_mail(self, receiver, server):
        '''
        This function sends the mail away.
        '''
        if self._verbose == 1:
            print "%s -> Try to send Mail, to -> \"%s\" ..." % (time.ctime(), receiver)

        # put together mail content
        subject = 'SRB LOG FILE PARSER NOTIFICATION - %s' % time.ctime(time.time())
        content = 'Hello,\n\nthis is an automatic generated mail from SRB LOG FILE PARSER [ Client ] ! Your are registered for recieving this notification for the SRB Server @ %s where between %s and %s -> %s interesting errors occured. \n\n---------------- error messages start ----------------\n\n' % (server, self._first_date, self._last_date, self._error_count)

        try:
            if self._error_count <= 5000:
                file_fd = open(self._mail_name, 'r')
                mail_error = file_fd.read()
                file_fd.close()
            else:
                mail_error = "!!!\n\nTo detailed error messages could not be supplied due to more than 5000 messages. Please check the database or the original SRB log file.\n\n!!!\n"

            if mail_error != "":
                content += mail_error
                content += '\n\n---------------- error messages end ----------------\n\nPlease do not respond to this mail!\n\n\nSRB LOG FILE PARSER [ CLIENT ]\n--\n[ powered by linux]'

                timus = time.strftime("%d %B %Y %H:%M:%S")

                text = 'From: '+self._smtp_from+'\n'
                text += 'To: '+receiver+'\n'
                text += 'Date: '+timus+'\n'
                text += 'Subject: '+subject+'\n'

                text = text + content

                # establish connection to smtp server
                server = smtplib.SMTP(self._smtp_server)
                server.login(self._smtp_user, self._smtp_pass)

                #transmit
                server.sendmail(self._smtp_from, receiver, text)
                #done
                if self._verbose == 1:
                    print "%s -> Mail sent to \"%s\" !" % (time.ctime(), receiver)
                server.quit()
                self._error_count = 0
                return 0
            else:
                if self._verbose == 1:
                    print "%s -> Nothing to send to \"%s\" !" % (time.ctime(), receiver)
                self._error_count = 0
                return -1
        except smtplib.SMTPAuthenticationError, e:
            if self._verbose == 1:
                print "%s -> Proplem with SMTP server authentication -> \"%s\" !" % (time.ctime(), e)
                print "\n"
            self._error_count = 0
            return -1
        except socket.error, e:
            if self._verbose == 1:
                print "%s -> Problem with SMTP server -> \"%s\" !" % (time.ctime(), e)
                print "\n"
            self._error_count = 0
            return -1
        except:
            if self._verbose == 1:
                print "%s -> Problem with sending mail to \"%s\" !" % (time.ctime(), receiver)
            self._error_count = 0
            return -1

    def delete_content(self):
        '''
        This function deletes the temorary file with the mail content.
        '''
        try:
            os.remove(self._mail_name)
            if self._verbose == 1:
                print "%s -> Deleted -> \"%s\"" % (time.ctime(), self._mail_name)
            return 0
        except OSError, e:
            if self._verbose == 1:
                print "%s -> Could not delete mail content file! -> \"%s\" -> %s" % (time.ctime(), self._mail_name, e)
            return -1

################# CLASS MyDatabase #################

class MyDatabase:
    '''
    This class deals with all the database issues.
    '''

    def __init__(self, error_description_file, error_desciption_path, databasename, database_path, serverlist, project, verbose):
        '''
        constructor
        '''
        self._verbose = verbose
        error = "%s/%s" % (error_desciption_path, error_description_file)
        self._db_access = None
        self._database_path = database_path
        #check if path exists
        if(1 == os.path.exists(database_path)):
            if self._verbose == 1:
                print "%s -> Database exists" % (time.ctime())# ----- debug ---
            os.chdir(database_path)
        else:
            #create wanted path
            if self._verbose == 1:
                print "%s -> Create database " % time.ctime() # --- debug ---
            os.mkdir(database_path)
            os.chdir(database_path)

        if(0 == os.path.exists(databasename)):
            try:
                # 1. create database
                self._connect = sqlite.connect(databasename, autocommit = 1)

                # 2. create access cursor
                self._db_access = self._connect.cursor()

                # 3. create tables

                sql = "CREATE TABLE error(e_id INTEGER NOT NULL PRIMARY KEY, e_number INT(10) NOT NULL, e_name CHAR(200) NOT NULL, e_description CHAR(400) NULL);"
                self._db_access.execute(sql)

                sql = "CREATE TABLE host (h_id INTEGER NOT NULL PRIMARY KEY, h_ip_address CHAR(15) NOT NULL, h_hostname CHAR(30) NULL);"
                self._db_access.execute(sql)

                sql = "CREATE TABLE host_project (hp_h_id INTEGER UNSIGNED NOT NULL, hp_p_id INTEGER UNSIGNED NOT NULL);"
                self._db_access.execute(sql)

                sql = "CREATE TABLE messages (m_id INTEGER NOT NULL PRIMARY KEY, m_date DATE NOT NULL, m_time TIME NOT NULL, m_error_string TEXT NOT NULL, m_line_number INT(7) NOT NULL, host_h_id INT(10) NOT NULL, error_e_id INT(10) NOT NULL);"
```

```
self._db_access.execute(sql)

sql = "CREATE TABLE project (p_id INTEGER NOT NULL PRIMARY KEY,
    p_name CHAR(100) NOT NULL);"
self._db_access.execute(sql)

# insert data if necessary
# insert error codes
error_file_fd = open(error, 'r')
content = error_file_fd.readline()              # get first line
x = 0
if self._verbose == 1:
    print "%s -> Initialising database ...\n" % time.ctime()
z = 0
while(1):
    if(content == "\n" or content == "\t"):
        content = error_file_fd.readline
    else:

        content = content.lstrip("{")       # remove first ”{”
        content_list = content.split(",")   # divide into pieces
        left = content_list[0].strip()      # remove whitespace
        if (left == '0' or left == '1'):    # remove non error codes
            content = error_file_fd.readline()
        else:
            if self._verbose == 1:
                x += 1
                # spinning line
                if (0 == x%2):
                    if z == 0:
                        sys.stdout.write("-\r")
                        sys.stdout.flush()
                        z = 1
                    elif z == 1:
                        sys.stdout.write("\\\r")
                        sys.stdout.flush()
                        z = 2
                    elif z == 2:
                        sys.stdout.write("|\r")
                        sys.stdout.flush()
                        z = 3
                    elif z == 3:
                        sys.stdout.write("/\r")
                        sys.stdout.flush()
                        z = 4
                    elif z == 4:
                        sys.stdout.write("-\r")
                        sys.stdout.flush()
                        z = 5
                    elif z == 5:
                        sys.stdout.write("\\\r")
                        sys.stdout.flush()
```

```
                                    z = 6
                                elif z == 6:
                                    sys.stdout.write("|\r")
                                    sys.stdout.flush()
                                    z = 7
                                elif z == 7:
                                    sys.stdout.write("/\r")
                                    sys.stdout.flush()
                                    z = 0
                        sys.stdout.flush
                        right = content_list[1].strip() # remove whitespace
                        sql = "INSERT INTO error (e_number, e_name) VALUES (%s, \"%s\");" % (left, right)
                        self._db_access.execute(sql)
                        content = error_file_fd.readline()
                        if(content == ''):
                            break

            error_file_fd.close()
            sql = "INSERT INTO error (e_number, e_name, e_description) VALUES (%s, \"%s\", \"%s\");" % (999999, "unknown", "unknown error number")
            self._db_access.execute(sql)

            #insert project
            sql = 'INSERT INTO project (p_name) VALUES ("%s")' % project
            self._db_access.execute(sql)

            for i in range(len(serverlist)):
                # insert in host
                sql = 'INSERT INTO host (h_ip_address) VALUES ("%s")' % serverlist[i][0]
                self._db_access.execute(sql)
                # get host id
                sql = 'SELECT * FROM host WHERE h_ip_address = "%s"' % serverlist[i][0]
                data = self._db_access.execute(sql)
                data = self._db_access.fetchall()
                host_id = data[0][0]
                # get project id
                sql = 'SELECT * FROM project WHERE p_name = "%s"' % project
                data = self._db_access.execute(sql)
                data = self._db_access.fetchall()
                project_id = data[0][0]
                # connect host and project
                sql = 'INSERT INTO host_project (hp_h_id, hp_p_id) VALUES (%s, %s)' % (host_id, project_id)
                data = self._db_access.execute(sql)

            if self._verbose == 1:
                print "\n%s -> Database new created !" % time.ctime()
        except:
            print "%s -> Problem creating database!" % time.ctime()
```

```
                os.rmdir(self._database_path)
                os._exit(-1)
        else:
            try:
                #check if tables there
                # 1. connect to  database
                self._connect = sqlite.connect(databasename, autocommit=1)

                # 2. create access cursor
                self._db_access = self._connect.cursor()

                # 3. check if table messages is still there
                sql = "SELECT * FROM messages"
                self._db_access.execute(sql)
                data = self._db_access.fetchall()
                if (0 == len(data)):
                    print "%s -> No data in table \"messages\" !" % (time.ctime())
                else:
                    print "%s -> Database holds %s error messages !" % (time.ctime(), len(data))

                # 4. check if table error is still there
                sql = "SELECT * FROM error"
                self._db_access.execute(sql)
                data = self._db_access.fetchall()
                if (0 == len(data)):
                    print "%s -> Database corruption detected: Missing data in table \"error\".\n\nIt's recommended to delete the database and initialise it again! It seems the original intialisation process was not completed.\n" % (time.ctime())
                else:
                    print "%s -> Database holds %s defined error numbers !" % (time.ctime(), len(data))

                # 5. check if table project is still there
                sql = "SELECT * FROM host_project"
                self._db_access.execute(sql)
                data = self._db_access.fetchall()
                if (0 == len(data)):
                    print "%s -> Database corruption detected: Missing connection between table \"host\" and \"project\".\n\nIt's recommended to delete the database and initialise it again! It seems the original intialisation process was not completed.\n" % (time.ctime())
                else:
                    print "%s -> Database holds %s defined connections between table \"project\" and \"host\" !" % (time.ctime(), len(data))

                # 6. check if table host_project is still there
                sql = "SELECT * FROM project"
                self._db_access.execute(sql)
                data = self._db_access.fetchall()
```

```python
        if (0 == len(data)):
            print "%s -> Database corruption detected: Missing project, insert new project \"%s\" into database!\n\nIt's recommended to delete the database and initialise it again! It seems the original intialisation process was not completed.\n" % (time.ctime(), project)
            #insert project
            sql = 'INSERT INTO project (p_name) VALUES ("%s")' % project
            self._db_access.execute(sql)
        else:
            print "%s -> Database holds %s defined projects !" % (time.ctime(), len(data))

        # 7. check if table host is still there
        sql = "SELECT * FROM host"
        self._db_access.execute(sql)
        data = self._db_access.fetchall()
        if (0 == len(data)):
            print "%s -> Database corruption detected: Missing data in table \"host\"\n\nIt's recommended to delete the database and initialise it again! It seems the original intialisation process was not completed.\n" % (time.ctime())
        else:
            print "%s -> Database holds %s defined hosts !" % (time.ctime(), len(data))

        # check if the is a new host in the log file
        for i in range(len(serverlist)):
            #check if host is there
            sql = 'SELECT * FROM host WHERE h_ip_address = "%s"' % serverlist[i][0]
            self._db_access.execute(sql)
            data = self._db_access.fetchall()
            if (0 == len(data)):
                if self._verbose == 1:
                    print "%s -> Insert new host \"%s\" into database !" % (time.ctime(), serverlist[i][0])
                # insert in host
                sql = 'INSERT INTO host (h_ip_address) VALUES ("%s")' % (serverlist[i][0])
                self._db_access.execute(sql)
                # get host id
                sql = 'SELECT * FROM host WHERE h_ip_address = "%s"' % serverlist[i][0]
                data = self._db_access.execute(sql)
                data = self._db_access.fetchall()
                host_id = data[0]["h_id"]
                # get project id
                sql = 'SELECT * FROM project WHERE p_name = "%s"' % project
                data = self._db_access.execute(sql)
                data = self._db_access.fetchall()
                project_id = data[0][0]
```

```
                        # connect host and project
                        sql = 'INSERT INTO host_project (hp_h_id, hp_p_id) VALUES (%s, %s)' % (host_id, project_id)
                        data = self._db_access.execute(sql)
            except sqlite.DatabaseError, e:
                print "%s -> %s" % (time.ctime(), e)
                os._exit(-1)

    def execute_sql(self, wait, database_obj, sql):
        '''
        This function tries to get access to a database for "wait" seconds. Either the sql query gets executed or the if no acccess is possible the program exits.
        '''
        for i in range(0, wait):
            try:
                database_obj.execute(sql)
                return 0, database_obj
            except sqlite.OperationalError:
                if self._verbose == 1:
                    if i%20 == 0:
                        text = "%s -> database temporary locked - keep trying for another %d seconds ...." % (time.ctime(), wait-i)
                        print text
                time.sleep(1)
            except:
                if self._verbose == 1:
                    print "%s -> database query execution error" % (time.ctime())

        return -1, database_obj

    def get_access_cursor(self):
        '''
        This function returns the database access cursor.
        '''
        return self._db_access

    def get_database_path(self):
        '''
        This function returns the database path
        '''
        return self._database_path

################# CLASS ClientThread #################

class ClientThread(threading.Thread):
    '''
    This class gets the information from the server and puts it into the database !
    '''
    def __init__(self, shared, db_object, address, port, cl_cert, cl_cert_path, ca_cert, ca_cert_path, verbose, mail_address, mail_ignore_error, smtp_server, smtp_pass, smtp_from, user, interval, filelist):
        '''
        Constructor
        '''

        self._file_list = filelist
        self._interval = interval
        self._verbose = verbose
        self._share = shared
        self._address = address
        self._port = port
        self._db_access = db_object
        self._client_certificate = cl_cert
        self._client_certificate_path = cl_cert_path
        self._client_ca = ca_cert
        self._client_ca_path = ca_cert_path
        threading.Thread.__init__(self)
        # create XML-reader
        self._xml_file_parser = make_parser()
        # turn off namespace
        self._xml_file_parser.setFeature(feature_namespaces, 0)
        self._smtp_password = smtp_pass
        self._mail_obj = []

        if self._smtp_password != None:
            for i in range(len(mail_address)):
                obj = Mail(mail_address[i], smtp_server, smtp_pass, smtp_from, user, verbose)
                self._mail_obj.append((obj, mail_address[i]))

            for i in range(len(self._mail_obj)):
                name = self._address+"-"+self._mail_obj[i][1]
                self._mail_obj[i][0].create_content(name)

        # overwrite the default ContextHandler with my own
        self._my_handler = MyContentHandler(self._db_access, self._address, mail_ignore_error, self._mail_obj, self._verbose )
        self._xml_file_parser.setContentHandler(self._my_handler)

        self._stop_thread = False # variable to indecate thread termination

    def run(self):
        '''
        This functions overwrites the standard run method.
        '''
        filenames = []

        if ( (0 == len(self._file_list)) & (self._stop_thread == False)):
            # if no old xml files fetch your own xml file
            try:
                if self._verbose == 1:
                    print "%s -> Client %d connecting to server %s" % (time.ctime(), thread.get_ident(), self._address)
                try:
                    #if self._verbose == 1:
                    #     print "try to connect: ",self._address
                    connect = self._connect_to_server(self._address, self._port)
                    #if self._verbose == 1:
                    #     print "connected -> ", connect
                except:
                    if self._verbose == 1:
                        print "%s -> Could not connect to host \"%s\"" % (time.ctime(), self._address)
                    if (self._smtp_password != None):
                        for g in range(len(self._mail_obj)):
                            self._mail_obj[g][0].delete_content()
                    self._stop_thread = True

                if (self._stop_thread == False):
                    # get file names
                    try:
                        if self._verbose == 1:
                            print "%s -> Get file names !!!" % time.ctime()
                        filenames = connect.get_file_list()
                        if ((-3 == filenames) & (self._stop_thread == False)):
                            # server is busy parsing
                            check = self._wait(connect)
                            if check == 0:
                                filenames = connect.get_file_list()
                                if (-3 == xml_content):
                                    # terminate thread
                                    self._stop_thread = True
                        if ((-2 == filenames) & (self._stop_thread == False)):
                            if self._verbose == 1:
                                print "%s -> RPC calls disabled !" % time.ctime()
                                self._stop_thread = True
                        if ((filenames == 0) & (self._stop_thread == False)):
                            filenames = []
                        if self._verbose == 1:
                            print "%s -> %s files to fetch " % (time.ctime(), len(filenames))
                    except:
                        if self._verbose == 1:
                            print "%s -> Could not connect (check) to IP  \"%s\"" % (time.ctime(), self._address)
                        if (self._smtp_password != None):
                            for g in range(len(self._mail_obj)):
                                self._mail_obj[g][0].delete_content()
                        self._stop_thread = True

                if ( (0 < len(filenames)) & (self._stop_thread == False)):
                    # fetch files
                    for g in range(len(filenames)):
                        xml_content = connect.get_my_xml_file(filenames[g])
                        if ( -3 == xml_content ):
```

```
                        if self._verbose == 1:
                            print "%s -> Parsing in progress ..." % time.ctime()
                        check = self._wait(connect)
                        if check == 0:
                            xml_content = connect.get_my_xml_file(filenames[g])
                            if (-3 == xml_content):
                                # terminate thread
                                self._stop_thread = True
                                break
                    if ( -2 == xml_content ):
                        if self._verbose == 1:
                            print "%s -> RPC calls disabled !" % time.ctime()
                        self._stop_thread = True
                        break
                    if (xml_content == "no file"):
                        # there is no new file available
                        if self._verbose == 1:
                            print "%s -> No file available !!!" % time.ctime()
                        self._stop_thread = True
                        break

                    # name of temporary XML file
                    name = "%s_client_xml_file_%d.xml" % (self._address, g)
                    # lock critical section
                    self._share.lock()
                    try:
                        c = g
                        while(1):
                            if(0 == os.path.exists(name)):
                                # save xml file locally
                                name = "%s_client_xml_file_%d.xml" % (self._address, c)
                                file_fd = open(name, 'w')
                                file_fd.write(xml_content)
                                file_fd.close()
                                self._file_list.append(name)
                                break
                            name = "%s_client_xml_file_%d.xml" % (self._address, c)
                            c += 1
                    finally:
                        # unlock critical section
                        self._share.release()
        except SSL.SSLError, e:
            if self._verbose == 1:
                print "%s -> Connection error (server \"%s\"): %s !" % (time.ctime(), self._address, e)
            self._stop_thread = True
        except:
            if self._verbose == 1:
                print "%s -> Error connecting to server -> \"%s\" !" % (time.ctime(), self._address)
            self._stop_thread = True

    if ( (0 < len(self._file_list)) & (self._stop_thread == False)):
        # deal with own generated file list
        for g in range(len(self._file_list)):
            name = self._file_list[g]
            if self._address == None:
                dbpath = "%s/"% self._db_access.get_database_path()
                ad = re.sub(dbpath, "", self._file_list[g])
                print ad
                ad = re.sub('_client_xml_file_[0-9]+.xml', "", ad)
                self._my_handler.set_ip(ad)
            try:
                file_fd = open(name, 'r')
            except IOError, e:
                print e
                self._stop_thread = True
                break # aborts for or while loop

            # write in database
            z = 0
            while(1):
                try:
                    if ((0 == self._share.set_variable(self)) & (self._stop_thread == False)):
                        try:
                            self._xml_file_parser.parse(file_fd)
                        except xml.sax.SAXParseException,e :
                            if self._verbose == 1:
                                print "%s -> sax parser error: %s" % ( time.ctime(), e)

                        self._share.reset_variable()
                        if (None != self._smtp_password):
                            # if mail is sendable
                            for a in range(len(self._mail_obj)):
                                self._mail_obj[a][0].send_mail(self._mail_obj[a][1], self._address)

                        file_fd.close()
                        os.remove(name)
                        if (self._smtp_password != None):
                            for g in range(len(self._mail_obj)):
                                self._mail_obj[g][0].delete_content()
                        break
                    else:
                        z += 1
                        if z == 10:
                            if self._verbose == 1:
                                print "%s -> can not access database --> terminating" % time.ctime()
                            break

                except AssertionError:
                    file_fd.close()
                    if self._verbose == 1:
                        print "%s -> can not access database -> terminating" % time.ctime()
                    if (self._smtp_password != None):
                        for g in range(len(self._mail_obj)):
                            self._mail_obj[g][0].delete_content()
                    break

                except:
                    file_fd.close()
                    if self._verbose == 1:
                        print "%s -> problem processing XML file -> terminating" % time.ctime()
                    if (self._smtp_password != None):
                        for g in range(len(self._mail_obj)):
                            self._mail_obj[g][0].delete_content()
                    break

    else:
        if (self._smtp_password != None):
            for g in range(len(self._mail_obj)):
                self._mail_obj[g][0].delete_content()
    print "%s -> client_thread %s STIRBT nun !!!" % (time.ctime(),thread.get_ident())

def _wait(self, connect_object):
    '''
    This function waits if the server is busy parsing the log file (busy waiting).
    '''
    counter = 0
    max_sleeping_time = 60 * self._interval
    sleeped = 0
    while(1):
        # check again
        check = connect_object.rpc_check_availabitity()
        if check == 0:
            return 0
        counter += 1
        if (counter == 30):
            if self._verbose == 1:
                print "%s -> Server takes a long time to parse file -> thread terminating" % time.ctime()# --- debug ---
            return -1
        # sleep for ten seconds and try again
        sleeped += 10
        if sleeped > max_sleeping_time:
            return -1
        time.sleep(10)

def _connect_to_server(self, server, port):
    '''
    This function establishes the connection to the server.
    '''
    serverus = server
    ctx = self.create_ctx()
    # connect to server via SSL using the created context
    urladdress = "https://%s:%d" % (serverus, port)
    server = Server(urladdress, SSL_Transport(ctx))
    # return server object
    return server

def create_ctx(self):
    '''
    The function creates the necessary SSL context using certificates.
    '''
    ctx = SSL.Context(protocol='sslv3') # use SSLv3 only
    ctx.load_cert(self._client_certificate_path+"/"+self._client_certificate)  # load client certificate
    ctx.load_client_CA(self._client_ca_path+"/"+self._client_ca)       # load certificate authority private key
   # if self._verbose == 1:
       # ctx.set_info_callback()          # tell me what you're doing —— debug ———
    ctx.set_session_id_ctx('server')    # session name
    return ctx

################# CLASS WorkerThread #################

class WorkerThread(threading.Thread):
'''
This class is responsible for starting the ClientThreads within a certain interval.
'''

def __init__(self, shared, db_object, interval, serverlist, cl_cert, cl_cert_path, ca_cert, ca_cert_path, verbose, mail_address, mail_ignore_error, smtp_server, smtp_pass, smtp_from, user):
    '''
    Constructor
    '''
    self._verbose = verbose
    self._share = shared
    self._db_access = db_object
    self._interval = interval
    self._serverlist = serverlist
    self._client_certificate = cl_cert
    self._client_certificate_path = cl_cert_path
    self._client_ca = ca_cert
    self._client_ca_path = ca_cert_path
    self._mail_address = mail_address
    self._mail_ignore_error = mail_ignore_error
    self._smtp_server = smtp_server
    self._smtp_pass = smtp_pass
    self._smtp_from = smtp_from
    self._smtp_user = user
    self._list = []
    threading.Thread.__init__(self)

def run(self):
    '''
    This function overwrites the standard run method.
    '''

    temp_list = []

    while(1):
        # deal with not processed, but fetched XML files first
        #find files
        os.path.walk(self._db_access.get_database_path(), self._parse_directory, self._list)
        if (0 < len(self._list)):
            temp_list = copy.deepcopy(self._list)
            self._thread = ClientThread(self._share, self._db_access, None, None, self._client_certificate, self._client_certificate_path, self._client_ca, self._client_ca_path, self._verbose, self._mail_address, self._mail_ignore_error, self._smtp_server, self._smtp_pass, self._smtp_from, self._smtp_user, self._interval, temp_list)
            self._thread.start()
            del self._list[:] # delete list content

        # then initiase new XML file fetching
        for i in range(len(self._serverlist)):
            dummy_list = []
            # start thread for fetching log file
            self._thread = ClientThread(self._share, self._db_access, self._serverlist[i][0], self._serverlist[i][1], self._client_certificate, self._client_certificate_path, self._client_ca, self._client_ca_path, self._verbose, self._mail_address, self._mail_ignore_error, self._smtp_server, self._smtp_pass, self._smtp_from, self._smtp_user, self._interval, dummy_list)
            self._thread.start()

        if self._verbose == 1:
            print "\n%s -> sleeping for %d minutes\n" % (time.ctime(), (self._interval))
        time.sleep(self._interval*60)


def _parse_directory(self, arg, dirname, fnames):
    '''
    This function "walks" through a given directory and considers all srbLOG*.gz files. The name and last modified time are saved in a list (2 dimensional array). The function should be used with os.path.walk(path, function_name, arg)!
    '''
    d = os.getcwd()
    # change into log file directory
    try:
        os.chdir(dirname)
    except:
        print "could not find directory \"%s\"" % dirname
        return -1
    # for each file
    for f in fnames:
        # check if file and if file is a log file e.g. srbLog.20051003.gz
        if (not os.path.isfile(f)) or (None == re.search('client_xml_file_[0-9]+.xml', f)):
            continue
        else:
        # save filename into an arrray (list)
            filus = dirname+"/"+f
            self._list.append(filus)
    # change back into the working directory
    os.chdir(d)


################## CLASS Mutex ##################

class Mutex:
'''
This class makes sure that only one client is writing into the database. This is necessary since sqlite is not trhead safe within a process! Futhermore is provide the possiblity to synchronise thread accessing critical sections.
'''
# database lock
_db_locked = threading.Lock()
# critical section lock
_locked = threading.Lock()

def __init__(self):
    '''
    Constructor
    '''
    self.writing = 0
    self._the_thread = 0

def set_variable(self, threadus):
    '''
    set variable writing
    '''
    Mutex._db_locked.acquire()  # lock
    # if nobody is accessing the database
    if self.writing == 0:
        #set variable
        self.writing = 1
        self._the_thread = threadus
        Mutex._db_locked.release()
        return 0
    else:
        if (1 != self._the_thread.isAlive()):
            # if the thread, which set the variable is dead, reset variable
            self.writing = 0
        Mutex._db_locked.release()   # release lock
        return -1

def reset_variable(self):
    '''
    reset variable writing
    '''
    Mutex._db_locked.acquire()  # lock
    self.writing = 0
    self._the_thread = 0
    Mutex._db_locked.release()

def lock(self):
    '''
    This functions acquires the look.
    '''
    Mutex._locked.acquire()

def release(self):
    '''
    This function releases the lock.
    '''
    Mutex._locked.release()
```

D.2.3 Module utils_client.py

LISTING D.7: Module utils_client.py

```
#!/usr/bin/env python
'''
This module provides basic funcitons for the client_classes.py and start_client.py.

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

import ConfigParser, string, os, sys, termios

```

```
def LoadConfig(file_name, config={}):
    """
    returns a dictionary with key's of the form
    <section>.<option> and the values

    source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/65334
    """
    config = config.copy()
    cp = ConfigParser.ConfigParser()
    cp.read(file_name)
    for sec in cp.sections():
        name = string.lower(sec)
        for opt in cp.options(sec):
            config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
    return config

def check_ip(ip):
    '''
    This function check if a given IP is valid.
    '''
    try:
        ip = ip.split(".")
    except AttributeError:
        return -1

    for i in range(len(ip)):
        check = ip[i].find("0", 0, 1)
        if -1 != check and 1 < len(ip[i]):
            return -1
        try:
            ip[i] = int(ip[i])
        except ValueError:
            return -1
        if ip[i] >= 0 and ip[i] <= 255:
            pass
        else:
            return -1

    return 0

def usage_exit(progname, msg=None):
    '''
    This function gives usage help and exits script.
    '''
    if msg:
        print msg
        print # lf cr
    print "usage: python %s [ -h|--help -c|--config -p|--smtp_passord -v|--verbose -d|--daemon] \n\n" % progname
    os._exit(-1)

def get_password(msg):
    '''
    This function reads from stdin without echoing the input.

    source: http://gnu.kookel.org/ftp/www.python.org/doc/faq/library.html
    modified by a. weise December 2005
    '''
    fd = sys.stdin.fileno()
    # turn off stdin's echoing
    old = termios.tcgetattr(fd)
    new = termios.tcgetattr(fd)
    new[3] = new[3] & ~termios.ICANON & ~termios.ECHO
    new[6][termios.VMIN] = 1
    new[6][termios.VTIME] = 0
    termios.tcsetattr(fd, termios.TCSANOW, new)
    s = ''     # save the characters typed and add them together
    try:
        print
        print msg
        while 1:
            c = os.read(fd, 1)
            if c == "\n":
                break
            s = s+c
    finally:
        # turn on stdin's echoing again
        termios.tcsetattr(fd, termios.TCSAFLUSH, old)
        return s
```

D.2.4 Script stop_client.sh

Listing D.8: Script stop_client.sh

```
#!/bin/sh
#
# Script to shutdown client daemon
#
# Reading University
# MSc in Network Centered Computing
# a.weise - a.weise@reading.ac.uk - December 2005
#
echo "stopping client ...."

name=start_client.py

# Find all clients
client_pid=`ps -elf | egrep $name | egrep -v grep | awk '{ print $4 }'`

#echo $client_pid

```

```
if [ "$client_pid" = "" ]
then
  echo No client is running !
else
  /bin/kill -15 $client_pid
      # sleep 3
  client_pid=`ps -elf | egrep $name | egrep -v grep | awk '{ print $4 }'`
  if [ "$client_pid" = "" ]
  then
    echo client stopped
  else
    /bin/kill -9 $client_pid
    echo client killed
  fi
fi
```

D.3 Virtualiser

D.3.1 Module gui.py

LISTING D.9: Module gui.py

```
#!/usr/bin/env python
'''
This Module is the start module for the display tool.

Reading University
MSc in Network Centred Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''
import gui_classes
import os, getopt, sys, re, time

# database
import sqlite

# functions
from gui_utils import usage_exit, check_date, convert_date, check_ip, find_item,
    help_context, check_time, LoadConfig

class Display:
    '''
    This is main class for the gui application.
    '''

    def __init__(self, config):
        '''
        Constructor
        '''
        workingpath = os.getcwd()

        self._database_name = config.get("database.name")
        self._database_path = config.get("database.path")
        self._database_path = self._database_path.rstrip("/")
        if(config.get("database.path") == '' or config.get("database.path") == None):
            # field is empty
            self._database_path = workingpath

        else:
            self._database_name = self._database_name.strip()
            if (-1 != self._database_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self._database_path = workingpath+"/"+self._database_path

        if(0 == os.access((self._database_path+"/"+self._database_name), 4)):     # 4 -> R_OK
            print "\nCould not access database under \"%s\" !\nMaybe change configuration file and try again!\n\n" % (self._database_path+"/"+self._database_name)
            os._exit(-1)

    def execute_sql(self, wait, database_obj, sql, col):
        '''
        This function tries to get access to a database for "wait" seconds. Either the sql query gets executed or the if no acccess is possible the program exits.
        '''

        for i in range(0, wait):
            try:
                database_obj.execute(sql)
                data = database_obj.fetchall()
                return data
            except sqlite.OperationalError:
                if i%10 == 0:
                    text = "database temporary locked - keep trying for another %d seconds ...." % (wait-i)
                    if col == 1:
                        col_obj = gui_classes.Colour()
                        text = col_obj.yellow(text)
                    print text
                time.sleep(1)

        text = "Database busy, could not apply request. Please try again."
        if col == 1:
            text = col_obj.yellow(text)

        print text,"\n\n"
        os._exit(0)

    def sql_host(self, col):
        '''
        This function gets all hosts from the database.
        '''
        sql = ' SELECT * FROM host, host_project, project'\
              ' WHERE host.h_id = host_project.hp_h_id '\
              ' AND host_project.hp_p_id = project.p_id'

        database = self._database_path+"/"+self._database_name

        connect = sqlite.connect(database, autocommit = 1)
        db_access = connect.cursor()
        data = self.execute_sql(120, db_access, sql, col)
        return data


    def sql_project(self, col):
        '''
        This function gets all projects from the database
        '''
        sql = ' SELECT * FROM project '
        database = self._database_path+"/"+self._database_name
        connect = sqlite.connect(database, autocommit = 1)
        db_access = connect.cursor()
        data = self.execute_sql(120, db_access, sql, col)
        return data


    def sql_error(self, col, d1 = None, d2 = None, t1 = None, t2 = None, host = None, project = None, error = None):
        '''
        This function gets all error messages from the database.
        '''
        where = 0
        sql = ' SELECT * FROM error, messages, host, host_project, project '

        if d1 != None and d2 != None:
            # between date1 and date2
            sql += ' WHERE '
            where = 1
            sql += ' messages.m_date BETWEEN "%s" AND "%s" ' % (d1, d2)

        elif d1 != None and d2 == None:
            # from date1 until now
            sql += ' WHERE '
            where = 1
            now = time.strftime("%Y-%m-%d", time.localtime())
            sql += ' messages.m_date BETWEEN "%s" AND "%s" ' % (d1, now)

        elif d1 == None and d2 != None:
            # until date2
            sql += ' WHERE '
            where = 1
            start_ = "1970-01-01"
            sql += ' messages.m_date BETWEEN "%s" AND "%s" ' % (start_, d2)

        else:
            pass

        if where == 0:
            sql += ' WHERE '
            where = 1
        else:
            sql += ' AND '

        if t1 != None and t2 != None:
            # between date1 and date2
            sql += ' messages.m_time BETWEEN "%s" AND "%s" ' % (t1, t2)
            sql += ' AND '

        elif t1 != None and t2 == None:
            # from date1 until now
            now = time.strftime("%H:%M:%S", time.localtime())
            sql += ' messages.m_time BETWEEN "%s" AND "%s" ' % (t1, now)
            sql += ' AND '

        elif t1 == None and t2 != None:
            # until date2
            start_ = "00:00:00"
            sql += ' messages.m_time BETWEEN "%s" AND "%s" ' % (start_, t2)
            sql += ' AND '

        sql += ' messages.error_e_id = error.e_id '

        if error != None:
            sql += ' AND ( '
            for i in range(len(error)):
                sql += ' error.e_number = \"%s\" ' % error[i]
                if len(error) > (i+1):
                    sql += ' OR '
            sql += ' ) '

        if host != None:
            sql += ' AND messages.host_h_id = host.h_id AND host.h_ip_address = "%s"
                ' % host
        elif host == None:
            sql += ' AND messages.host_h_id = host.h_id '

        sql += ' AND host.h_id = host_project.hp_h_id  '\
              ' AND host_project.hp_p_id = project.p_id '

        if project != None:
            sql += ' AND project.p_name = "%s" ' % project


        sql += 'ORDER BY messages.m_date, messages.m_time'

        database = self._database_path+"/"+self._database_name
        connect = sqlite.connect(database, autocommit = 1)
        db_access = connect.cursor()
        data = self.execute_sql(120, db_access, sql, col)
        return data

    def display_graph(self, dataset, col, file_fd = None):
        '''
        This function displays a barchart diagram containing Error Numbers and the corresponding Frequency
        '''
        field = []
        field_label = []
        table_error = []
        for i in range(len(dataset)):
            # prepare data
            index = find_item(int(dataset[i]['error.e_number']), field)
            if (None == index):
                field.append([int(dataset[i]['error.e_number']), 1])
                field_label.append([int(dataset[i]['error.e_number']), 1])
                table_error.append( [int(dataset[i]['error.e_number']), 1, dataset[i]['error.e_name']])
            else:
                count = field[index][1]
                count += 1
                field[index][1] = count
                field_label[index][1] = count
                table_error[index][1] = count

        field.sort()
        field_label.sort()
        table_error.sort()

        h_line = "-----------------------------------------------------------------------------------"
        v_line = "|"
        header = "\nFrequency of Errors: \n"

        if file_fd != None:
            content = header
            content += "\n\n Nr.    | Error Number\t| Frequency\t| Error Name\t\t\t\n\n"
            file_fd.write(content)

        if col == 1:
            col_obj = gui_classes.Colour()
            header = col_obj.yellow(header)
            h_line = col_obj.yellow(h_line)
            v_line = col_obj.yellow(v_line)

        print header
        print h_line
        print " Nr.    "+v_line+" Error Number\t"+v_line+" Frequency\t"+v_line+" Error Name\t\t\t"
        print h_line

        for i in range(len(table_error)):
            print " %5d %s %7s\t%s %6s\t%s %s" % ((i+1), v_line, table_error[i][0], v_line, table_error[i][1], v_line, table_error[i][2])

            if file_fd != None:
                content = " %5d | %7s\t| %6s\t| %s\n" % ((i+1), table_error[i][0], table_error[i][1], table_error[i][2])
                file_fd.write(content)

        print h_line

        for i in range(len(field)):
            field_label[i][0] = (i+1)
            field_label[i][1] = "%d" % field[i][0]
            field[i][0] = (i+1)

        window = []

        pic_obj = gui_classes.Picture(col, window)
        pic_obj.show_barchart("Diagram \"Error Number - Frequency\"", field, field_label, "ERROR NUMBER", "FREQUENCY", dataset, select_type = "error", filus_fd = file_fd, descript = "Diagram \"Error Number - Frequency\"")
        pic_obj.mainloop()

########################################################################

def start():

    '''
    Start the application.
    '''
    verbose = 0
    col = 1
    graph = 0
    filus = None
    configfile = ""
    sql_host = None
    sql_project = None
    sql_error = None
    sql_error_freq = None
    date1 = None
    date2 = None
    time1 = None
    time2 = None
    ip = None
    port = None
    project = None
    error = None

    # evaluate parameters
    try:
        opts, args = getopt.getopt(sys.argv[1:], 'c:vhg', ['config=', 'verbose', '
            graph', 'nocolor', 'help', 'sql_host', 'sql_project', 'sql_error', '
            sql_error_freq', 'start_date=', 'end_date=', 'start_time=', 'end_time=',
            'ip=', 'port=', 'project=', 'error=' , 'file='])
        for opt, value in opts:
            if opt in ('', '--nocolor'):
                col = 0
            if opt in ('-h','--help'):
                msg = help_context(col)
                usage_exit(sys.argv[0], msg, col)
            if opt in ('-c', '--config'):
                value = value.replace("=", "")
                configfile = os.getcwd()+"/"+value
            if opt in ('-v', '--verbose'):
                verbose = 1
            if opt in ('-g', '--graph'):
                graph = 1

        for opt, value in opts:
            if opt in ('', '--sql_host'):
                sql_host = 1
            if opt in ('', '--sql_project'):
                sql_project = 1
            if opt in ('', '--sql_error'):
                sql_error = 1
            if opt in ('', '--sql_error_freq'):
                sql_error_freq = 1
            if opt in ('', '--error'):
                error = value
                error = error.strip()
                error = error.strip(",")
                error = error.split(",")
                for i in range(len(error)):
                    error[i] = error[i].strip()
                    try:
                        error[i] = int(error[i])
                    except ValueError, e:
                        #'given error is not valid'
                        usage_exit(sys.argv[0], 'invalid literal for int()' , col)
            if opt in ('', '--start_date'):
                date1 = value
                status = re.search('^[0-3][0-9].[0-1][0-9].[1-9][0-9]{3}', date1)
                if (None == status):
                    usage_exit(sys.argv[0], 'given date is not valid', col)
                else:
                    date1 = status.string[status.start():status.end()]
                    if (0 == check_date(date1)):
                        date1 = convert_date(date1)
                    else:
                        usage_exit(sys.argv[0], 'given date is not valid', col)
            if opt in ('','--end_date'):
                date2 = value
                status = re.search('^[0-3][0-9].[0-1][0-9].[1-9][0-9]{3}', date2)
                if (None == status):
                    usage_exit(sys.argv[0], 'given date is not valid', col)
                else:
                    date2 = status.string[status.start():status.end()]
                    if (0 == check_date(date2)):
                        date2 = convert_date(date2)
                        print "date 2: ", date2
                    else:
                        usage_exit(sys.argv[0], 'given date is not valid', col)
            if opt in ('', '--start_time'):
                time1 = value
                status = re.search('^[0-2][0-9]:[0-5][0-9]:[0-5][0-9]', time1)
                if (None == status):
                    usage_exit(sys.argv[0], 'given time is not valid', col)
                else:
                    time1 = status.string[status.start():status.end()]
                    if ( 0 == check_time(time1)):
                        pass
                    else:
                        usage_exit(sys.argv[0], 'given time is not valid', col)
            if opt in ('', '--end_time'):
                time2 = value
                status = re.search('^[0-2][0-9]:[0-5][0-9]:[0-5][0-9]', time2)
                if (None == status):
                    usage_exit(sys.argv[0], 'given time is not valid', col)
                else:
                    time2 = status.string[status.start():status.end()]
                    if ( 0 == check_time(time2)):
                        pass
                    else:
                        usage_exit(sys.argv[0], 'given time is not valid', col)
            if opt in ('','--ip'):
                ip = value
                status = re.search('^[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}', ip)
                if (None == status):
                    usage_exit(sys.argv[0], 'given IP is not valid', col)
                else:
                    ip = status.string[status.start():status.end()]
                    if (0 == check_ip(ip)):
                        print "ip: ", ip
                    else:
                        usage_exit(sys.argv[0], 'given IP is not valid', col)
            if opt in ('', '--port'):
                port = int(value)
                if (port < 1024 or port > 50001):
                    usage_exit(sys.argv[0], "Server port is out of range! \nMake sure the server port lies between 1025 (inclusive) and 50000 (inclusive)!\n\n", col)
            if opt in ('','--project'):
                project = value
            if opt in  ('', '--file'):
                filus = value
    except getopt.error, e:
        e = "%s" % (e)
        usage_exit(sys.argv[0], e, col)
    except ValueError, e:
        e = "%s" % (e)
        usage_exit(sys.argv[0], e, col)


    # load config file or default values
    if (configfile != ""):
        # check if file exists
        if(1 == os.path.exists(configfile)):
            config = LoadConfig(configfile)
        else:
            # if file NOT exists terminate program
            print "\n\nSorry, a given config file does NOT exist !\nPlease try again !\n\n"
            os._exit(-1)
    else:
        msg = "\nNo config file spezified !\n"
        usage_exit(sys.argv[0], msg, col)

    if col == 1:
        col_obj = gui_classes.Colour()

    gui = Display(config)

    if verbose == 1:
        i = 1
        d = config.iteritems()
        while(1):
            try:
                print i, ". ", d.next()
                i += 1
            except:
                break

    if filus != None:
        # save output in file
        # check if file exists
```

```
    try:
        filus_fd = file(filus, 'r')
        quest = "File \"%s\" already exists, overwrite file (y/n) ? -> " % filus
        if col == 1:
            quest = col_obj.darkred(quest)
        decision = raw_input(quest)
        if decision == 'y' or decision == 'Y' or decision == 'yes' or decision == 'Yes' or decision == 'YES':
            filus_fd.close()
            filus_fd = file(filus, 'w')
        else:
            os._exit(0)
    except IOError:
        filus_fd = file(filus, 'w')

#------------------------------------------- SQL COMMANDS -------------------------------------------#

if (1 == sql_host):
    print "sql_host: ", sql_host
    data = gui.sql_host(col)

    if len(data) == 0:
        text = "\n\nSorry, no data available for you request!"
        if col == 1:
            text = col_obj.yellow(text)
        print text
        print "\n\n"
        os._exit(0)

    h_line = "--------------------------------------------------------------------------------"
    v_line = "|"

    if col == 1:
        h_line = col_obj.yellow(h_line)
        v_line = col_obj.yellow(v_line)

    # table head
    print h_line
    print " Nr.\t"+v_line+"\tHost IP\t\t\t"+v_line+"      Host Name       "+v_line+"      Project "
    print h_line

    # table data
    for i in range(len(data)):
        print " %d\t%s\t%s\t\t%s %17s  %s  %s" % ((i+1), v_line, data[i]['host.h_ip_address'], v_line, data[i]['host.h_hostname'], v_line, data[i]['project.p_name'])

    print h_line
elif (1 == sql_project):
    print "sql_project: ", sql_project
    data = gui.sql_project(col)

    if len(data) == 0:
        text = "\n\nSorry, no data available for you request!"
        if col == 1:
            text = col_obj.yellow(text)
        print text
        print "\n\n"
        os._exit(0)

    h_line = "-----------------------------------------"
    v_line = "|"

    if col == 1:
        h_line = col_obj.yellow(h_line)
        v_line = col_obj.yellow(v_line)

    # table head
    print h_line
    print v_line+" Nr.\t"+v_line+"\tProject\t\t\t"+v_line
    print h_line

    # table data
    for i in range(len(data)):
        print "%s %d\t%s\t%s\t\t%s" % (v_line, (i+1), v_line, data[i]['p_name'],
            v_line)

    print h_line


elif (1 == sql_error):

    data = gui.sql_error(col, d1=date1, d2=date2, t1 = time1, t2 = time2, host =
        ip, project = project, error = error)

    if len(data) == 0:
        text = "\n\nSorry, no data available for you request!"
        if col == 1:
            text = col_obj.yellow(text)
        print text
        print "\n\n"
        os._exit(0)

    # table head
    h_line = "
        ---------------------------------------------------------------------------------------------
        "
    v_line = "|"
    h_line_short = "-"
    header = "  Nr.  |    Date    |   Time   |\t\t\t\tError String"
    ln = "LN"
    en = "EN"
    ip = "IP"
    pr = "PR"

    brown_line = h_line
    if col == 1:
        brown_line = col_obj.brown(h_line)
        h_line = col_obj.yellow(h_line)
        v_line = col_obj.yellow(v_line)
        h_line_short = col_obj.yellow(h_line_short)
        header = col_obj.yellow(header)
        ln = col_obj.yellow(ln)
        en = col_obj.yellow(en)
        ip = col_obj.yellow(ip)
        pr = col_obj.yellow(pr)

    print h_line

    print header

    if filus != None:
        content = " Nr.\t|    Date    |   Time   | \t\t\t\tError String\t\t\t\t | Line Number | Error Number |     Host IP     | Project\n\n"
        filus_fd.write(content)
    print h_line

    # table data console
    for i in range(len(data)):
        print "%6d %s %10s %s %6s %s %70s" % ((i+1), v_line, data[i]['messages.m_date'], v_line, data[i]['messages.m_time'], v_line, data[i]['messages.m_error_string'])
        print "%s: %7s %s %s: %6s %s %s: %15s %s %s: %s" % (ln, data[i]['messages.m_line_number'], h_line_short, en, data[i]['error.e_number'], h_line_short, ip, data[i]['host.h_ip_address'], h_line_short, pr, data[i]['project.p_name'])

        # table data file
        if filus != None:
            content = " %d\t| %10s | %6s | %70s | %10s  | %10s   | %15s | %s \n" % ((i+1), data[i]['messages.m_date'], data[i]['messages.m_time'], data[i]['messages.m_error_string'], data[i]['messages.m_line_number'], data[i]['error.e_number'], data[i]['host.h_ip_address'], data[i]['project.p_name'])
            filus_fd.write(content)
        if len(data) > (i+1):
            print brown_line


    print h_line

    print "\nAbbreviations:\n\n%s - Line Number in original SRB log file\n%s - Error Number\n%s - Host IP Address\n%s - Project\n\n" % (ln, en, ip, pr)

    if graph == 1:
        if filus != None:
            gui.display_graph(data, col, file_fd = filus_fd)
        else:
            if filus != None:
                filus_fd.close()
            gui.display_graph(data, col)

elif ( 1 == sql_error_freq):

    data = gui.sql_error(col, d1=date1, d2=date2, t1 = time1, t2 = time2, host = ip, project = project, error = error)

    if len(data) == 0:
        text = "\n\nSorry, no data available for you request!"
        if col == 1:
            text = col_obj.yellow(text)
        print text
        print "\n\n"
        os._exit(0)


    print "datasets: ", len(data)
    #print data
    if graph == 0:
        field = []
        field_label = []
        table_error = []
        for i in range(len(data)):
            index = find_item(int(data[i]['error.e_number']), field)
            if (None == index):
                field.append([int(data[i]['error.e_number']), 1])
                field_label.append([int(data[i]['error.e_number']), 1])
                table_error.append( [int(data[i]['error.e_number']), 1, data[i]['error.e_name']])
              # print field
            else:
                count = field[index][1]
                count += 1
                field[index][1] = count
                field_label[index][1] = count
                table_error[index][1] = count

        field.sort()
        field_label.sort()
        table_error.sort()

        h_line = "----------------------------------------------------------------------
```

```
                "
            v_line = "|"
            header = "\nFrequency of Errors: \n"

            if col == 1:
                header = col_obj.yellow(header)
                h_line = col_obj.yellow(h_line)
                v_line = col_obj.yellow(v_line)

            print header
            print h_line
            print " Nr.   "+v_line+" Error Number\t"+v_line+" Frequency\t"+v_line+" Error Name\t\t\t"
            print h_line

            if filus != None:
                content = " Nr.   | Error Number\t| Frequency\t| Error Name\t\t\t\n\n"
                filus_fd.write(content)

            # print table console
            for i in range(len(table_error)):
                print " %5d %s %7s\t%s %6s\t%s %s" % ((i+1), v_line, table_error[i][0], v_line, table_error[i][1], v_line, table_error[i][2])

                # print table in file
                if filus != None:
                    content = " %5d | %7s\t| %6s\t| %s" % ((i+1), table_error[i][0], table_error[i][1], table_error[i][2])
                    filus_fd.write(content)

            print h_line

            if filus != None:
                filus_fd.close()

        elif graph == 1:
            if filus != None:
                gui.display_graph(data, col, file_fd = filus_fd)
            else:
                gui.display_graph(data, col)

if __name__ == '__main__':

    start()
```

D.3.2 Module gui_classes.py

LISTING D.10: Module gui_classes.py

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

'''
This module contains the classes for the display tool. It is needed by the module "
    gui.py".

Reading University
MSc in Network Centred Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

from gui_utils import find_item, complete_days, complete_hours, complete_ticks
import Tkinter
import tkFileDialog
import tkMessageBox
import Graphs
import tooltips
from gui_utils import second, second_string_to_int, second_string_only


class Colour:
    '''
    This class uses the ANSI escape sequences to color the output !
    '''
    color = {"reset":"\x1b[0m",
            "bold":"\x1b[01m",
            "teal":"\x1b[36;06m",
            "turquoise":"\x1b[36;01m",
            "fuscia":"\x1b[35;01m",
            "purple":"\x1b[35;06m",
            "blue":"\x1b[34;01m",
            "darkblue":"\x1b[34;06m",
            "green":"\x1b[32;01m",
            "darkgreen":"\x1b[32;06m",
            "yellow":"\x1b[33;01m",
            "brown":"\x1b[33;06m",
            "red":"\x1b[31;01m",
            "darkred":"\x1b[31;06m"}

    def __init__(self):
        '''
        Constructor
        '''
        pass

    def green(self, text):
        '''
        dye green
        '''
        return self.color['green']+text+self.color['reset']

    def red(self, text):
        '''
        dye red
        '''
        return self.color['red']+text+self.color['reset']

    def bold(self, text):
        '''
        dye bold
        '''
        return self.color['bold']+text+self.color['reset']

    def teal(self, text):
        '''
        dye teal
        '''
        return self.color['teal']+text+self.color['reset']

    def turquoise(self, text):
        '''
        dye turquoise
        '''
        return self.color['turquoise']+text+self.color['reset']

    def fuscia(self, text):
        '''
        dye fuscia
        '''
        return self.color['fuscia']+text+self.color['reset']

    def purple(self, text):
        '''
        dye purple
        '''
        return self.color['purple']+text+self.color['reset']

    def darkred(self, text):
        '''
        dye darkred
        '''
        return self.color['darkred']+text+self.color['reset']

    def darkblue(self, text):
        '''
        dye darkblue
        '''
        return self.color['darkblue']+text+self.color['reset']

    def blue(self, text):
        '''
        dye blue
        '''
        return self.color['blue']+text+self.color['reset']

    def darkgreen(self, text):
        '''
        dye darkgreen
        '''
        return self.color['darkgreen']+text+self.color['reset']

    def yellow(self, text):
        '''
        dye yellow
        '''
        return self.color['yellow']+text+self.color['reset']

    def brown(self, text):
        '''
        dye brown
        '''
        return self.color['brown']+text+self.color['reset']


class Picture(Tkinter.Tk):
    '''
    This class provides functions around the "diplay diagrams" issues.
    '''

    def __init__(self, color, windows):
        '''
        Constructor
        '''
        self._col = color
        self._windows = []
        # needed to close all windows properly
        self._all_windows = windows
        # needed for deactivate and activate all window buttons properly
        self._all_windows.append(self)
        # initialise tkinter
        Tkinter.Tk.__init__(self)
        # set min size
        self.minsize(width=500, height=400)
        #create frame, where the diagram is drawn later
        self.framus = Tkinter.Frame(self)
        # add frame to dialog
        self.framus.grid(
            column = 0,
            row = 0,
            columnspan = 7,
            sticky = "news" #north east west south
            )
        # "QUIT" BUTTON
        self.button_quit = Tkinter.Button(self, text="quit")
        self.button_quit.grid(
            column = 6,
            row = 1,
            columnspan = 1,
            sticky = "e"
                         )
        # tooltips for "QUIT" button
        tooltips.ToolTip(self.button_quit, follow_mouse=1, text="Please press \"quit\" to close this window. Note, all windows, which are opened from this window (child windows) are closed as well !", delay=3500)
        self.button_quit.configure(command = self.pre_shutdown)
        # "SAVE AS" BUTTON
        self.button_save = Tkinter.Button(self, text = "save as")
        self.button_save.grid(
            column = 5,
            row = 1,
            columnspan = 1,
            sticky = "e"
                        )
        self.button_save.configure(command = self.save_as)
        # status bar
        self.status = Tkinter.Label(self)
        self.status.grid(
            column = 0,
            row = 3,
            columnspan = 7,
            sticky = "w"
                        )
        # configure grid
        self.grid_columnconfigure(0, weight = 1)
        self.grid_rowconfigure(0, weight = 1)

        # overwrite function
        self.protocol("WM_DELETE_WINDOW", self.shutdown)

        # saves as button hoover method
        self.button_save.bind("<Enter>", self._show_save_as_description)
        self.button_save.bind("<Leave>", self._hide_description)
        # tooltips for "SAVE AS" button
        tooltips.ToolTip(self.button_save, follow_mouse = 1, text = "Please press \"save as\" to save the diagram as a postscript file.", delay = 3500)

    def deactivate(self):
        '''
        This function deactivates all buttons.
        '''
        self.protocol("WM_DELETE_WINDOW", self._dummy)
        self.button_quit.configure(command = self._dummy)
        self.button_save.configure(command = self._dummy)
  if self._select_type == 'error' or self._select_type == 'date':
            self.button_select.configure(command = self._dummy)

    def activate(self):
        '''
        This function activates all buttons.
        '''
        self.protocol("WM_DELETE_WINDOW", self.shutdown)
        self.button_quit.configure(command = self.pre_shutdown)
        self.button_save.configure(command = self.save_as)
        if self._select_type == 'error':
            self.button_select.configure(command = self._select_error)
        elif self._select_type == 'date':
            self.button_select.configure(command = self._select_date)

    def save_as(self):
        '''
        This function saves the diagram picture as postscript.
        '''
        # deactivate all buttons
        try:
            for i in range(len(self._all_windows)):
                self._all_windows[i].deactivate()
        except Tkinter.TclError:
            pass
        # save as dialog
        result = tkFileDialog.asksaveasfilename(filetypes = [('postscript', '*.ps')], title = 'Save graph as ... ')
        # activate all buttons
        try:
            for i in range(len(self._all_windows)):
                self._all_windows[i].activate()
        except Tkinter.TclError:
            pass

        if result != '':
            # save diagram in file
            self.graph.canvas.postscript(file = result, colormode = 'color')
            ###Graphs.canvas.postscript(file = result, colormode = 'color')

    def _dummy(self, event = None):
        '''
        This function is doing nothing, it serves as a dummy.
        '''
        return 'break'

    def show_barchart(self, window_name, listus, label, xlabel, ylabel,  data, select_type=None, filus_fd = None, descript = None):
        '''
        This function shows a barchart diagram.
        '''
        self.title(window_name)
        self._data = data
        self._file_fd = filus_fd
self._select_type = select_type
```

```
        # generate barchart diagram
        line = Graphs.GraphBars(listus , color ='green', size = 6)
        graphObject = Graphs.GraphObjects([line])
        self.graph = Graphs.GraphBase(self.framus, 400, 400, relief = 'sunken', border = 2, listerus = label, x_label = xlabel, y_label = ylabel, header = window_name, description = descript, label_interval = 10)
        self.graph.pack(side = Tkinter.LEFT, fill = Tkinter.BOTH, expand = Tkinter.YES)
        self.graph.draw(graphObject, 'automatic', 'automatic')

        # sort items for listbox
        self.items = []
        self.search_label = []
        self.search_value = []
        for i in range(len(label)):
            self.items.append(label[i][1])
            self.search_value.append(listus[i])
            self.search_label.append(label[i])

        self.create_listbox()

    def show_line(self, window_name, listus, label, xlabel, ylabel, data, select_type = None, error = None, labelamount = 10, filus_fd = None, descript = None, typ = None):
        '''
        This functions shows a line chart diagram.

        window_name = name of the new window
        listus = value list
        label = label list for x-axis
        xlabel = description of x-axis
        ylabel = description of y-axis
        select_type = type of items are listed in listbox
        data = dataset which comes from the database query
        error = chosen error from listbox
        labelamount = amount of possible labels for the x-axis
        '''
        self.title(window_name)
        self._data = data
        self._file_fd = filus_fd
  self._select_type = select_type

        values = []

        # only draw a dot where is a real value
        for i in range(len(listus)):
            if listus[i][1] != 0 :
                values.append(listus[i])

        dot = Graphs.GraphSymbols(values, color = 'green', marker = 'dot', fillcolor = 'darkgreen')

        if len(listus) > 1:
            line = Graphs.GraphLine(listus, color='green', size=6)
            graphObject = Graphs.GraphObjects([line, dot])
        else:
            graphObject = Graphs.GraphObjects([dot])

        self.graph = Graphs.GraphBase(self.framus, 600, 400, relief = 'sunken', border = 2, listerus = label, x_label = xlabel, y_label = ylabel, header = window_name, description = descript, label_interval = labelamount, type = typ)
        self.graph.pack(side = Tkinter.LEFT, fill = Tkinter.BOTH, expand = Tkinter.YES)
        self.graph.draw(graphObject, 'automatic', 'automatic')

        if select_type == "date":

            self.items = []
            self.search_label = []
            self.search_value = []
            #rearrange labels for listbox
            for i in range(len(label)):
                if listus[i][1] != 0 :
                    self.items.append(label[i][1])
                    self.search_value.append(listus[i])
                    self.search_label.append(label[i])

            # create listbox with dates
            self.create_listbox(error)

    def create_listbox(self, error = None):
        '''
        This function creates a listbox with the given items.
        '''
         # listbox
        list_scrollbar = Tkinter.Scrollbar(self, orient=Tkinter.VERTICAL)
        list_scrollbar.grid ( row = 1, column = 1, columnspan = 1, sticky = "ns" )

        self.listbox = Tkinter.Listbox(self, height = 4, cursor = "plus", bg = "#ffffff", bd = 1, highlightcolor = "#00ff00", yscrollcommand=list_scrollbar.set)
        self.listbox.grid(
            column = 0,
            row = 1,
            columnspan = 1,
            sticky = "news"
                         )

        self.listbox.bind("<Enter>", self._show_description)
        self.listbox.bind("<Leave>", self._hide_description)

        list_scrollbar["command"] = self.listbox.yview

        # ”PLOT” button
        self.button_select = Tkinter.Button(self, text = "plot")
        self.button_select.grid(
            column = 3,
            row = 1,
            columnspan = 1,
            sticky = "w"
                        )

        self._the_error = error

        if self._select_type == 'error':
            self.button_select.configure(command = self._select_error)
        elif self._select_type == 'date':
            self.button_select.configure(command = self._select_date)

        self.button_select.bind("<Enter>", self._show_plot_description)
        self.button_select.bind("<Leave>", self._hide_description)
        # tooltips for ”PLOT” button
        tooltips.ToolTip(self.button_select, follow_mouse = 1, text = "Please press \"plot\" to generate a new diagram with the selected item from the listbox.", delay = 3500)

        # OPTION (dropdown) menu
        if self._select_type == 'error':
            self._ldate = "%15s" % ("error")
            tooltips.ToolTip(self.listbox, follow_mouse = 1, text = "Please select an error and then press \"plot\" to view this error number only.")
            tooltips.ToolTip(self.listbox, follow_mouse = 1, text = "Please select an error and then press \"plot\" to view this error number only.")
        elif self._select_type == 'date':
            self._ldate = "%15s" % ("date")
            tooltips.ToolTip(self.listbox, follow_mouse = 1, text = "Please select a date and then press \"plot\" to view this date only.")
        self._lfreq = "%12s" % ("frequency")
        self.var = Tkinter.StringVar(self)
        # activate a trace, which monitores the changes, so in case the drop down menu is used a function is called
        self.var.trace('w', self.menu_change)
        self.var.set(self._ldate) # initial value

        option = Tkinter.OptionMenu(self, self.var, self._ldate, self._lfreq)

        option.bind("<Enter>", self._show_dropdown_description)
        option.bind("<Leave>", self._hide_description)

        if self._select_type == 'error':
            tooltips.ToolTip(option, follow_mouse = 1, text = "Select \"error\" or \"frequency\" to change the order in the listbox:\nerror -> order by error numbers (ascending)\nfrequency -> order by frequency (ascending).")
        elif self._select_type == 'date':
            tooltips.ToolTip(option, follow_mouse = 1, text = "Select \"date\" or \"frequency\" to change the order in the listbox:\ndate -> order by dates (ascending)\nfrequency -> order by frequency (ascending).")

        option.grid(
            column = 2,
            row = 1,
            columnspan = 1,
            sticky = "w"
                   )

        # SPACE LABEL
        labelus = Tkinter.Label(self)
        labelus.grid(
            column = 4,
            row = 1,
            columnspan = 1,
            sticky = "news"
                    )

    def menu_change(self, name, index, mode):
        '''
        This function changes the order in the listbox according to the chosen item in the drop down menu.
        '''
        temp_listus = []
        temp_search_label = []

        change = self.var.get()
        # for dates
        if change == self._ldate:

            if self._select_type == 'error':
                self.search_label.sort(second_string_to_int)
                self._dropdown_description = "change order in listbox, currently ordered by \"error number\""
            elif self._select_type == 'date':
                self.search_label.sort(second_string_only)
                self._dropdown_description = "change order in listbox, currently ordered by \"date\""


            for i in range(len(self.search_label)):
                # save label
                temp = self.search_label[i][1]
                 # search for corresponding label in label array
                for j in range(len(self.search_value)):
                    if self.search_label[i][0] == self.search_value[j][0]:
                        # save corresponding label
                        temp_listus.append([i+1, self.search_value[j][1]])
                # adjust items
                temp_search_label.append([i+1, temp])
```

```python

                self.items[i] = "%s  (%s)" % (temp_search_label[len(temp_search_label)-1][1], temp_listus[len(temp_listus)-1][1])

            self.search_value = temp_listus[:]
            self.search_label = temp_search_label[:]

            # delete old items and write new items in listbox
            self.listbox.delete(0, Tkinter.END)
            for i in range(len(self.items)):
                self.listbox.insert(Tkinter.END, self.items[i])

        # for frequency
        elif change == self._lfreq:

            self._dropdown_description = "change order in listbox, currently ordered by \"frequency\""
            self.search_value.sort(second)

            # rearrange order of array
            for i in range(len(self.search_value)):
                #save value
                temp = self.search_value[i][1]
                # search for corresponding label in label array
                for j in range(len(self.search_label)):
                    if self.search_label[j][0] == self.search_value[i][0]:
                        # save corresponding label
                        self.items[i] = "%s  (%s)" % (self.search_label[j][1], self.search_value[i][1])
                        temp_search_label.append([i+1, self.search_label[j][1]])
                # adjust number in value array
                self.search_value[i][0] = i+1
                self.search_value[i][1] = temp

            # rearrange label array description
            self.search_label = temp_search_label[:]

            # delete old items and write new items in listbox
            self.listbox.delete(0, Tkinter.END)
            for i in range(len(self.items)):
                self.listbox.insert(Tkinter.END, self.items[i])

    def pre_shutdown(self):
        '''
        This function calls a message box and make sure the user really wants to shutdown.
        '''
        # deactivate all buttons
        try:
            for i in range(len(self._all_windows)):
                self._all_windows[i].deactivate()
        except Tkinter.TclError:
            pass
        # queston message box
        status = tkMessageBox.askquestion("Close Window", "Do you really want to close this and all child windows ?")
        # activate all buttons
        try:
            for i in range(len(self._all_windows)):
                self._all_windows[i].activate()
        except Tkinter.TclError:
            pass
        if status == 'yes':
            self.shutdown()

    def shutdown(self):
        '''
        This function closes all open child windows and itself
        '''

        if self._file_fd != None:
            if self._all_windows[0] == self:
                # only main windows closes file
                self._file_fd.close()

        # destroy all cildren windows
        for i in range(len(self._windows)):
            try:
                self._windows[i].shutdown()

            except Tkinter.TclError:
                pass

        # destroy myself
        try:
            self.destroy()
        except Tkinter.TclError:
            pass

    def _select_error(self):
        '''
        This function get the selected item from the listbox
        '''
        try:
            # get index of chosen listbox item
            firstIndex = self.listbox.curselection()[0]
        except IndexError:
            firstIndex = None

        if firstIndex != None:

            # convert index to int
            firstIndex = int(firstIndex)

            # print data
            title = "Diagram Error %s \"Frequency - Date\"" % self.search_label[firstIndex][1]

            field = []
            field_label = []
            data_new = []
            # work up the given data and prepare for display
            for i in range(len(self._data)):
                if (int(self._data[i]['error.e_number']) == int(self.search_label[firstIndex][1])):
                    data_new.append(self._data[i]) # get new dataset (only interesting data is taken)
                    index = find_item(self._data[i]['messages.m_date'], field)
                    if (None == index):
                        field.append([self._data[i]['messages.m_date'], 1])
                        field_label.append([self._data[i]['messages.m_date'], 1])
                    else:
                        count = field[index][1]
                        count += 1
                        field[index][1] = count
                        field_label[index][1] = count

            field.sort()
            field_label.sort()

            # print result table
            h_line = "-----------------------------------"
            v_line = "|"
            header = "\nFrequency of Error \"%s\":\n" % self.search_label[firstIndex][1]

            # write in file
            if self._file_fd != None:
                content =  "\n"+header
                content += "\n\n Nr.    | Date\t\t| Frequency\n\n"
                self._file_fd.write(content)

            if self._col == 1:
                col_obj = Colour()

                header = col_obj.yellow(header)
                h_line = col_obj.yellow(h_line)
                v_line = col_obj.yellow(v_line)

            print header
            print h_line
            print " Nr.    "+v_line+" Date\t\t"+v_line+" Frequency"
            print h_line

            for i in range(len(field)):
                print " %5d %s %s\t%s %s" % ((i+1), v_line, field[i][0], v_line, field[i][1])

                # write in file
                if self._file_fd != None:
                    content = " %5d | %s\t| %s\n" % ((i+1),  field[i][0], field[i][1])
                    self._file_fd.write(content)

            print h_line

            for i in range(len(field_label)):
                temp = field_label[i][0]
                field_label[i][0] = field_label[i][1]
                field_label[i][1] = temp

            for i in range(len(field)):
                field_label[i][0] = (i+1)
                field_label[i][1] = "%s" % field[i][0]
                field[i][0] = (i+1)

            field_label, field = complete_days(field_label, field)

            pic_obj = Picture(self._col, self._all_windows)

            self._windows.append(pic_obj)

            title = "Diagram \"Frequency - Date\" - Error: %s" % self.search_label[firstIndex][1]
            descr = title+" - Range: "+field_label[0][1]+"  -  "+field_label[len(field_label)-1][1]#+" )"
            pic_obj.show_line(title, field, field_label, "DATE", "FREQUENCY", data_new, select_type = "date", error = self.search_label[firstIndex][1], labelamount = 10, filus_fd = self._file_fd, descript = descr, typ = "date" )

            pic_obj.mainloop()

        else:
            # disable all buttons within the windows
            for i in range(len(self._all_windows)):
                self._all_windows[i].deactivate()

            tkMessageBox.showerror("Error", "No item selected !")
            # enable all buttons within the windows
            for i in range(len(self._all_windows)):
                self._all_windows[i].activate()

    def _select_date(self):
        '''
        This functions take a date and generates a new graph
        '''

        try:
            firstIndex = self.listbox.curselection()[0]
        except IndexError:
            firstIndex = None

        if firstIndex != None:

            firstIndex = int(firstIndex)

            # print data
            title = "Diagram \"Frequency - Time\" - Date %s" % self.search_label[firstIndex][1]

            field = []
            field_label = []
            data_new = []
            for i in range(len(self._data)):
                if self._data[i]['messages.m_date'] == self.search_label[firstIndex][1] and int(self._the_error) == int(self._data[i]['error.e_number']):

                    data_new.append(self._data[i])
                    hour = self._data[i]['messages.m_time'].split(":")

                    hour[0] = int(hour[0])# hour

                    index = find_item(hour[0], field)
                    if (None == index):
                        field.append([hour[0], 1])
                        field_label.append([hour[0], 1])
                    else:
                        count = field[index][1]
                        count += 1
                        field[index][1] = count
                        field_label[index][1] = count

            field.sort()
            field_label.sort()

            # rearrange arrays for use within the picture and graph class
            for i in range(len(field_label)):
                temp = field_label[i][0]
                field_label[i][0] = field_label[i][1]
                field_label[i][1] = temp

            for i in range(len(field)):
                field_label[i][0] = (i+1)
                field_label[i][1] = "%s" % field[i][0]
                field[i][0] = (i+1)

            field_label, field = complete_hours(field_label, field)
            field_label, field = complete_ticks(field_label, field)

            h_line = "-------------------------------"
            v_line = "|"
            header = "\nFrequency on Date \"%s\":\n" % self.search_label[firstIndex][1]

            # write in file
            if self._file_fd != None:
                content = "\n"+header
                content += "\n\n Time of Day\t| Frequency\n\n"
                self._file_fd.write(content)

            if self._col == 1:
                col_obj = Colour()
                print col_obj.yellow(header)
                h_line = col_obj.yellow(h_line)
                v_line = col_obj.yellow(v_line)

            print h_line
            print " Time of Day\t"+v_line+" Frequency"
            print h_line

            for i in range(len(field)):
                print " %2s h - %2s h\t%s %s" % (i, i+1, v_line, field[i][1])

                if self._file_fd != None:
                    content = " %2s h - %2s h\t| %s\n" % (i , i+1, field[i][1])
                    self._file_fd.write(content)

            print h_line

            pic_obj = Picture(self._col, self._all_windows)
            self._windows.append(pic_obj)

            pic_obj.show_line(title, field, field_label, "TIME OF DAY (hrs)", "FREQUENCY", data_new , select_type = "time", labelamount=24, filus_fd = self._file_fd, descript = title )
            pic_obj.mainloop()

        else:

            for i in range(len(self._all_windows)):
                self._all_windows[i].deactivate()
            tkMessageBox.showerror("Error", "No item selected !")
            for i in range(len(self._all_windows)):
                self._all_windows[i].activate()


    def _show_description(self, event):
        '''
        This function displays the description for the listbox in the status bar.
        '''
        if self._select_type == 'error':
            self.status.config(text = "listbox: error number (frequency) -> select error to zoom", anchor = "w")
        if self._select_type == 'date':
            self.status.config(text = "listbox: date (frequency) for the choosen error -> select date to zoom", anchor = "w")
        self.status.update_idletasks()

    def _show_plot_description(self, event):
        '''
        This function displays the description of the "plot" button in the status bar.
        '''
        self.status.config(text = "plot new diagram", anchor = "w")
        self.status.update_idletasks()

    def _hide_description(self, event):
        '''
        This function deletes the status bar content.
        '''
        self.status.config(text="")
        self.status.update_idletasks()

    def _show_save_as_description(self, event):
        '''
        This function show the description of the "save as" button in the status bar.
        '''
        self.status.config(text = "save diagram as postscript file", anchor = "w")
        self.status.update_idletasks()

    def _show_dropdown_description(self, event):
        '''
        This function shows a short description for the dropdown menu in the status bar.
        '''

        self.status.config(text = self._dropdown_description, anchor = "w")
        self.status.update_idletasks()
```

D.3.3 Module gui_utils.py

Listing D.11: Module gui_utils.py

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

'''
This module provides small utility methods that are used by the gui_classes.py and gui.py.
```

```

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''
import os, time, ConfigParser, string
import gui_classes
import calendar

def LoadConfig(file_name, config={}):
    """
    returns a dictionary with key's of the form
    <section>.<option> and the values

    source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/65334
    """
    config = config.copy()
    cp = ConfigParser.ConfigParser()
    cp.read(file_name)
    for sec in cp.sections():
        name = string.lower(sec)
        for opt in cp.options(sec):
            config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
    return config

def usage_exit(progname, msg = None, color = 0):
    '''
    This function gives usage help and exits script.
    '''
    if msg:
        if 1 == color and msg != None:
            color_obj = gui_classes.Colour()
            print color_obj.red(msg)
        else:
            print msg
        print # lf cr

    text = "usage: python %s -c config_file [optional commands] \n\n" % progname
    if 1 == color:
        print color_obj.red(text)
    else:
        print text
    os._exit(-1)

def check_time(timus):
    '''
    This functions checks if a given time with the format hour:minute:second (12:45:46) is valid.
    '''
    timus = timus.split(':')
    if int(timus[0]) < 24 and int(timus[1]) < 60 and int(timus[2]) < 60:
        return 0
```

```python
    else:
        return -1

def check_date(datus):
    '''
    This function checks if a given date is valid.
    '''
    datus = datus.split(".")
    tup1 = (int(datus[2]), int(datus[1]), int(datus[0]), 0, 0, 0, 0, 0, 0)
    try:
        date = time.mktime (tup1)
        tup2 = time.localtime (date)
        if tup1[:2] != tup2[:2]:
            return -1
        else:
            return 0
    except OverflowError:
        return -1
    except ValueError:
        return -1

def convert_date(datus):
    '''
    This function converts a date like 01.10.2005 into database conform date like 2005-10-01.
    '''
    datus = datus.split(".")
    return "%s-%s-%s" % (datus[2], datus[1], datus[0])

def convert_date_readable(datus):
    '''
    This function converts a date like 2005-10-10 into are readable format 01.10.2005.
    '''
    datus = datus.split("-")
    return "%s.%s.%s" % (datus[2], datus[1], datus[0])

def check_ip(ip):
    '''
    This function checks if a given IP is valid.
    '''
    try:
        ip = ip.split(".")
    except AttributeError:
        return -1

    for i in range(len(ip)):
        check =  ip[i].find("0", 0, 1)
        if -1 != check and 1 < len(ip[i]):
            return -1
        try:
            ip[i] = int(ip[i])
        except ValueError:
            return -1
        if ip[i] >= 0 and ip[i] <= 255:
            pass
        else:
            return -1

    return 0

def find_item(search, listus):
    '''
    This function find an item within a list (2 dimensional)
    '''
    for i in range(len(listus)):
        if 1 == len(listus[i]):
            if listus[i] == search:
                return i
        elif 2 == len(listus[i]):
            if listus[i][0] == search:
                return i
        elif 3 == len(listus[i]):
            if listus[i][0][0] == search:
                return i
    return None

def help_context(color):
    '''
    This function provides the help context.
    '''

    color_obj = gui_classes.Colour()
    msg = ''
    if color == 1:
        msg += color_obj.green("\n-----------   Help   ----------\n\n\n")
    else:
        msg += "\n-----------   Help   ----------\n\n\n"

    note = "PLEASE NOTE, if you give parameter values, please do not enter characters like \" \" (space) or \"!\", because this could be characters which are interpreted by the terminal. If you have to enter such characters, please escape them like \"\!\".\n\n"
    if color == 1:
        msg += color_obj.purple(note)
    else:
        msg += note

    msg += "-c or --config\t\t\t-> defines config file, if no config file given, default values are used\n"\
        "-v or --verbose\t\t\t-> activates printing of messages [debug option]\n"\
        "-h or --help\t\t\t-> print this help\n"\
        "-g or --graph\t\t\t-> show output additionally as a diagram\n"\
        "--nocolor\t\t\t-> no colored console output\n"\
        "--file <string>\t\t\t-> dump output into a file (file name has to be given)\n"

    if color == 1:
        msg += color_obj.green("\n-----  database commands  -----\n\n")
    else:
        msg += "\n-----  database commands  -----\n\n"
    msg += "--sql_host\t\t\t-> show all hosts\n"\
        "--sql_project\t\t\t-> show all projects\n"\
        "--sql_error\t\t\t-> show errors (additional parameters possible)\n"\
        "--sql_error_freq\t\t-> show only frequency of errors (additional parameters possible)\n"
    if color == 1:
        msg += color_obj.green("\n-----  additional parameters  -----\n")
    else:
        msg += "\n-----  additional parameters  -----\n"
    msg += "\n--start_date <date>\t\t-> start date (e.g. 23.12.2005)\n"\
        "--end_date <date>\t\t-> end date (e.g. 23.01.2006)\n"\
        "--start_time <time>\t\t-> start time (e.g. 23:12:19)\n"\
        "--end_time <time>\t\t-> end time (e.g. 23:12:59)\n"\
        "--ip <ip>\t\t\t-> host IP (e.g. 127.0.0.1)\n"\
        "--project <string>\t\t-> specify a certain project\n"\
        "--error <int,int...>\t\t-> specify a certain error (comma seperated list)\n"
    if color == 1:
        msg += color_obj.green("\n-----  examples  -----\n\n")
        msg += color_obj.blue("python gui.py -c config_gui.ini --sql_project\n")
        msg += color_obj.yellow("\t-> show all projects\n\n")

        msg += color_obj.blue("python gui.py -c config_gui.ini --sql_host\n")
        msg += color_obj.yellow("\t-> show all host and the corresponding project\n\n")

        msg += color_obj.blue("python gui.py -c config_gui.ini --sql_error --start_date 01.01.2005 --end_date 01.03.2005 --ip 127.0.0.1\n")
        msg += color_obj.yellow("\t-> show all errors of localhost between 01.01.2005 and 01.03.2005\n\n")

        msg += color_obj.blue("python gui.py -c config_gui.ini --sql_error --start_date 01.01.2005  --project mySRBproject\n")
        msg += color_obj.yellow("\t-> show all errors between 01.01.2005 and now for the project \"mySRBproject\"\n\n")

        msg += color_obj.blue("python gui.py -c config_gui.ini --sql_error --start_date 22.10.2005 --end_date 22.10.2005\n--start_time 12:00:00 --end_time 18:00:00  --ip 127.0.0.1 --file test.txt\n")
        msg += color_obj.yellow("\t-> show all errors on the 22.10.2005 between 12 h and 18 h on localhost and\n\t    save output in file \"test.txt\"\n\n")

        msg += color_obj.blue("python gui.py -c config_gui.ini --sql_error_freq --error -1023 --ip 127.0.0.1 -g \n")
        msg += color_obj.yellow("\t-> show error frequency for the error -1023 from host 127.0.0.1 and display diagram\n\n")


    else:
        msg += "\n-----  examples  -----\n\n"
        msg += "python gui.py -c config_gui.ini --sql_project\n"
        msg += "\t-> show all projects\n\n"

        msg += "python gui.py -c config_gui.ini --sql_host\n"
        msg += "\t-> show all host and the corresponding project\n\n"

        msg += "python gui.py -c config_gui.ini --sql_error --start_date 01.01.2005 --end_date 01.03.2005 --ip 127.0.0.1\n\t-> show all errors of localhost between 01.01.2005 and 01.03.2005\n\n"

        msg += "python gui.py -c config_gui.ini --sql_error --start_date 01.01.2005 --project mySRBproject\n"
        msg += "\t-> show all errors between 01.01.2005 and now for the project \"mySRBproject\"\n\n"

        msg += "python gui.py -c config_gui.ini --sql_error --start_date 22.10.2005 --end_date 22.10.2005\n--start_time 12:00:00 --end_time 18:00:00  --ip 127.0.0.1 --file test.txt\n"
        msg += "\t-> show all errors on the 22.10.2005 between 12 h and 18 h on localhost and\n\t   save output in file \"test.txt\"\n\n"

        msg += "python gui.py -c config_gui.ini --sql_error_freq --error -1023 --ip 127.0.0.1 -g \n"
        msg += "\t-> show error frequency for the error -1023 from host 127.0.0.1 and display diagram\n\n"

    msg += "\n"

    return msg

def complete_hours(label, field):
    '''
    This function completes the missing hours within an array
    '''
    new_hours = []
    new_values = []

    count = 0
    for i in range(len(label)):
        temp = "%d" % count
        while(count < 24):
            if temp == label[i][1]:
                break
            new_hours.append([count, temp])
            new_values.append([count, 0])
            count += 1
            temp = "%d" % count

        new_hours.append([count, label[i][1]])
        new_values.append([count, field[i][1]])

        count += 1

    # last hours
    while(count <= 23):
        temp = "%d" % count
        new_hours.append([count, temp])
        new_values.append([count, 0])
        count += 1

    return new_hours, new_values

def complete_days(days, value_field):
    '''
    This function completes the missing dates within an array.
    '''
    if len(days) == 1:
        # if only one day in the field
        return days, value_field

    new_days = [] # new array with the completed days
    new_values = []

    for i in range(len(days)):
        day1 = days[i][1].split("-")
        day2 = days[i+1][1].split("-")

        for x in range(len(day1)):
            day1[x] = int(day1[x])
            day2[x] = int(day2[x])

        if day1[0] == day2[0] and day1[1] == day2[1] and (day1[2]+1) == day2[2]:
            # save day1 in new array
            temp1 = "%d" % day1[2]
            temp2 = "%d" % day1[1]
            date = "%d-" % day1[0]
            if len(temp2) == 1:
                date += "0%d-" % day1[1]
            else:
                date += "%d-" % day1[1]

            if len(temp1) == 1:
                date += "0%d" % (day1[2])
            else:
                date += "%d" % (day1[2])

            if len(new_days) == 0:
                number = 1
            else:
                number = int(new_days[len(new_days)-1][0])+1
            new_days.append([number, date])
            new_values.append([number, value_field[i][1]])
        else:
            # not the following day
            if day1[0] == day2[0] and day1[1] == day2[1]:
                #year and month the same
                new_days, new_values = complete_d(new_days, day1, day2, new_values, value_field[i][1])

            elif day1[0] == day2[0]:
                # year the same
                new_days, new_values = complete_m(new_days, day1, day2, new_values, value_field[i][1])

            else:
                # year change
                new_days, new_values = complete_y(new_days, day1, day2, new_values, value_field[i][1])

        if (i+2) == len(days):
            break

    # add last date
    temp1 = "%d" % day2[2]
    temp2 = "%d" % day2[1]
    date = "%d-" % day2[0]

    if len(temp2) == 1:
        date += "0%d-" % day2[1]
    else:
        date += "%d-" % day2[1]

    if len(temp1) == 1:
        date += "0%d" % day2[2]
    else:
        date += "%d" % day2[2]

    if len(new_days) == 0:
        number = 1
    else:
        number = int(new_days[len(new_days)-1][0])+1
    new_days.append([number, date])
    new_values.append([number, value_field[i+1][1]])

    return new_days, new_values

def complete_d(new_array, start_date, end_date, new_field, value):
    '''
    Add missing dates within a month
    '''

    month = calendar.monthcalendar(start_date[0], start_date[1])
    # run through matrix and save all dates between day1 and day2 in new_days array
    found = 0
    terminate = 0
    for x in range(len(month)):
        if terminate != 0:
            break
        for y in range(len(month[x])):
            # go to current day1
            if terminate != 0:
                break
            if start_date[2] == month[x][y] and found == 0:
                #save date1 in new_days
                temp1 = "%d" % start_date[2]
                temp2 = "%d" % start_date[1]

                date = "%d-" % start_date[0]

                if len(temp2) == 1:
                    date += "0%d-" % start_date[1]
                else:
                    date += "%d-" % start_date[1]

                if len(temp1) == 1:
                    date += "0%d" % start_date[2]
                else:
                    date += "%d" % start_date[2]

                if (0 < len(new_array)):
                    number = int(new_array[len(new_array) -1][0])+1
                else:
                    number = 0
                new_array.append([number, date])
                new_field.append([number, value])

                found = 1
            elif found == 1:
                # add new dates
                if end_date[2] == month[x][y]:
                    terminate = 1
                else:
                    # save dates
                    temp1 = "%d" %  month[x][y]
                    temp2 = "%d" % end_date[1]

                    date = "%d-" % end_date[0]

                    if len(temp2) == 1:
                        date += "0%d-" % end_date[1]
                    else:
                        date += "%d-" % end_date[1]

                    if len(temp1) == 1:
                        date += "0%d" % month[x][y]
                    else:
                        date += "%d" % month[x][y]

                    # get next entry number in arrays
                    if (0 < len(new_array)):
                        number = int(new_array[len(new_array) -1][0])+1
                    else:
                        number = 0
                    new_array.append([number, date])
                    new_field.append([number, 0])
            else:
                pass

    return new_array, new_field

def complete_m(new_array, start_date, end_date, new_field, value):
    '''
    This function adds missing dates within a year.
    '''
    start_month = start_date[1]
    end_month = end_date[1]

    #current month
    month = calendar.monthrange(start_date[0], start_date[1])
    temp_date2 = [start_date[0], start_date[1], month[1]]

    new_array, new_field = complete_d(new_array, start_date, temp_date2, new_field, value)

    start_month += 1

    # month in between
    while(start_month < end_month):

        month = calendar.monthrange(start_date[0], start_month)
        temp_date2 = [start_date[0], start_month, month[1]]
        temp_date1 = [start_date[0], start_month, 1]

        new_array, new_field = complete_d(new_array, temp_date1, temp_date2, new_field, 0)

        start_month += 1

    # last month
    temp_date1 = [start_date[0], start_month, 1]

    new_array, new_field = complete_d(new_array, temp_date1, end_date, new_field, 0)

    return new_array, new_field

def complete_y(new_array, start_date, end_date, new_field, value):
    '''
    This function adds missing dates within many years
    '''
    start_year = start_date[0]
    end_year = end_date[0]

    # current year
    temp_date2 = [start_date[0], 12, 31]
    new_array, new_field = complete_m(new_array, start_date, temp_date2, new_field,
        value)

    start_year += 1

    # years in between
    while(start_year < end_year):

        temp_date1 = [start_year, 1, 1]
        temp_date2 = [start_year, 12, 31]
        new_array, new_field = complete_m(new_array, temp_date1, temp_date2,
            new_field, 0)
        start_year += 1

    # last year
    temp_date1 = [start_year, 1, 1]
    new_array, new_field = complete_m(new_array, temp_date1, end_date, new_field, 0)

    return new_array, new_field

def complete_ticks(label, values):
    '''
    This function adds bins, so that the dot in the time diagram are between two
        hours.
    '''
    new_label = []
    new_values = []

    count = 0
    for i in range(2*len(label)):
        if (i%2) != 0:
            new_values.append([i, values[count][1]])
            new_label.append([i, ""])
            count += 1
        else:
            new_label.append([i, label[count][1]])

    return new_label, new_values

def second(t1, t2 ):
    '''
    This function works with sort and the field gets sorted descending, but the
        second value within the array is taking into account !!!
    '''
    # sort descending
    return t2[1] - t1[1]

def second_string_to_int(t1, t2):
    '''
    This function works with sort and the field gets sorted ascending, but the second value within the array is taking into account !!! (The values to be sort are number as strings.)
    '''
    # sort ascending
    return int(t1[1]) - int(t2[1])

def second_string_only(t1, t2):
    '''
    This function works with sort and the field gets sorted ascending, but the second value within the array is taking into account !!! (The values to be sort are strings.)
    '''
    # sort ascending
    return cmp(t1[1], t2[1])
```

D.4 Remote Controller

Listing D.12: Module admin_server.py

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

'''
This module can be used to administer the server (daemon).

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

# config parsing
import ConfigParser, string

#misc
import os, getopt, sys, re

# connection issues
from M2Crypto.m2xmlrpclib import Server, SSL_Transport
from M2Crypto import SSL

def LoadConfig(file, config={}):
    """
    This function returns a dictionary with key's of the form
    <section>.<option> and the corresponding values.

    source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/65334
    """
    config = config.copy()
    cp = ConfigParser.ConfigParser()
    cp.read(file)
    for sec in cp.sections():
        name = string.lower(sec)
        for opt in cp.options(sec):
            config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
    return config


class Colour:
    '''
    This class uses the ANSI escape sequences to color the output !
    '''
    color = {"reset":"\x1b[0m",
             "bold":"\x1b[01m",
             "teal":"\x1b[36;06m",
             "turquoise":"\x1b[36;01m",
             "fuscia":"\x1b[35;01m",
             "purple":"\x1b[35;06m",
             "blue":"\x1b[34;01m",
             "darkblue":"\x1b[34;06m",
             "green":"\x1b[32;01m",
             "darkgreen":"\x1b[32;06m",
             "yellow":"\x1b[33;01m",
             "brown":"\x1b[33;06m",
             "red":"\x1b[31;01m",
             "darkred":"\x1b[31;06m"}

    def __init__(self):
        '''
        Constructor
        '''
        pass

    def green(self, text):
        '''
        dye green
        '''
        return self.color['green']+text+self.color['reset']

    def red(self, text):
        '''
        dye red
        '''
        return self.color['red']+text+self.color['reset']

    def bold(self, text):
        '''
        dye bold
        '''
        return self.color['bold']+text+self.color['reset']

    def teal(self, text):
        '''
        dye teal
        '''
        return self.color['teal']+text+self.color['reset']

    def turquoise(self, text):
        '''
        dye turquoise
        '''
        return self.color['turquoise']+text+self.color['reset']

    def fuscia(self, text):
        '''
        dye fuscia
        '''
        return self.color['fuscia']+text+self.color['reset']

    def purple(self, text):
        '''
        dye purple
        '''
        return self.color['purple']+text+self.color['reset']

    def darkred(self, text):
        '''
        dye darkred
        '''
        return self.color['darkred']+text+self.color['reset']

    def darkblue(self, text):
        '''
        dye darkblue
        '''
        return self.color['darkblue']+text+self.color['reset']

    def blue(self, text):
        '''
        dye blue
        '''
        return self.color['blue']+text+self.color['reset']

    def darkgreen(self, text):
        '''
        dye darkgreen
        '''
        return self.color['darkgreen']+text+self.color['reset']
    def yellow(self, text):
        '''
        dye yellow
        '''
        return self.color['yellow']+text+self.color['reset']

    def brown(self, text):
        '''
        dye brown
        '''
        return self.color['brown']+text+self.color['reset']


class Admin:
    '''
    This is manager class for the Remote Controller application.
    '''

    def __init__(self, config):
        '''
        Constructor
        '''
        workingpath = os.getcwd()

        # varify user input
        self.__client_certificate = config.get("files.client_certificate")
        self.__client_certificate_path = config.get("path.path_client_certificate")
        self.__client_certificate_path = self.__client_certificate_path.rstrip("/")
        if(config.get("path.path_client_certificate") == '' or config.get("path.
            path_client_certificate") == None):
            self.__client_certificate_path = workingpath
        else:
            self.__client_certificate = self.__client_certificate.strip()
            if (-1 != self.__client_certificate_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self.__client_certificate_path = workingpath+"/"+self.
                    __client_certificate_path

        self.__client_ca = config.get("files.client_ca")
        self.__client_ca_path = config.get("path.path_client_ca")
        self.__client_ca_path = self.__client_ca_path.rstrip("/")
        if(config.get("path.path_client_ca") == '' or config.get("path.path_client_ca
            ") == None):
            self.__client_ca_path = workingpath
        else:
            self.__client_ca = self.__client_ca.strip()
            if (-1 != self.__client_ca_path.find("/", 0, 1)):
                # first character "/"
                pass
            else:
                self.__client_ca_path = workingpath+"/"+self.__client_ca_path

        # check if file are existing
        if(0 == os.access((self.__client_ca_path+"/"+self.__client_ca), 4)):     # 4 R_OK
            print "\nCould not access client ca certificate under \"%s\" !\nMaybe change configuration file and try again!\n\n" % (self.__client_ca_path+"/"+self.__client_ca)
            os._exit(-1)

        if(0 == os.access((self.__client_certificate_path+"/"+self.__client_certificate), 4)):     # 4 R_OK
            print "\nCould not access client certificate under %s !\nMaybe change configuration file and try again!\n\n" % (self.__client_certificate_path+"/"+self.__client_certificate)
            os._exit(-1)

    def connect_to_server(self, server, port):
        '''
        This function establishs the connection to the server.
        '''
        serverus = server

        ctx = self.create_ctx()
        # connect to server via SSL using the created context
        urladdress = "https://%s:%d" % (serverus, port)
        server = Server(urladdress, SSL_Transport(ctx))
        # return server object
        return server

    def create_ctx(self):
        '''
        This funciton creates the SSL context to establish an encrypted connetion by using certificates.
        '''
        ctx = SSL.Context(protocol='sslv3') # use SSLv3 only
        ctx.load_cert(self.__client_certificate_path+"/"+self.__client_certificate)            # load client certificate
        ctx.load_client_CA(self.__client_ca_path+"/"+self.__client_ca)          # load certificate authority private key
        # ctx.set_info_callback()              # tell me what you're doing --- debug -----
        ctx.set_session_id_ctx('server')      # session name
        return ctx

# ---------------- additional functions -----------------

def usage_exit(progname, msg = None, color = 1):
    '''
    This function gives usage help and exits the module.
    '''
    if msg:
        if 1 == color and msg != None:
            color_obj = Colour()
            print color_obj.red(msg)
        else:
            print msg
        print # lf cr

    text = "usage: python %s -c config_file [optional commands] \n\n" % progname
    if 1 == color:
        print color_obj.red(text)
    else:
        print text
    os._exit(-1)

def check_ip(ip):
    '''
    This function checks if a given IP is valid and returns -1 for an invalid IP
        address otherwise 0.
    '''

    try:
        ip = ip.split(".") # split in 4 number
    except AttributeError:
        return -1

    for i in range(len(ip)):
        check =  ip[i].find("0", 0, 1)
        if -1 != check and 1 < len(ip[i]):
            return -1
        try:
            ip[i] = int(ip[i])
        except ValueError:
            return -1
        if ip[i] >= 0 and ip[i] <= 255: # check if number is between 0 and 255
            pass
        else:
            return -1

    return 0

def find_item(search, listus):
    '''
    This function finds an item within a list (1-3 dimensional) and returns the list
        index otherwise "None".
    '''
    for i in range(len(listus)):
        if 1 == len(listus[i]):
            if listus[i] == search:
                return i
        elif 2 == len(listus[i]):
            if listus[i][0] == search:
                return i
        elif 3 == len(listus[i]):
            if listus[i][0][0] == search:
                return i
    return None

def help_context(color):
    '''
    This function provides the help context.
    '''

    color_obj = Colour()
    msg = ''
    if color == 1:
        msg += color_obj.green("\n-----------   Help   ----------\n\n\n")
    else:
        msg += "\n-----------   Help   ----------\n\n\n"

    note = "PLEASE NOTE, if you give parameter values, please do not enter characters like \" \" (space) or \"!\", because this could be characters which are interpreted by the terminal. If you have to enter such characters, please escape them like \"\!\".\n\n"
    if color == 1:
        msg += color_obj.purple(note)
    else:
        msg += note

    msg += "-c or --config\t\t\t-> defines config file, if no config file given, default values are used\n"\
        "-h or --help\t\t\t-> print this help\n"\
        "--nocolor\t\t\t-> no colored console output\n"
    if color == 1:
        msg += color_obj.green("\n-----  server commands  -----\n\n")
    else:
        msg += "\n-----  server commands  -----\n\n"
    msg += "--rpc_status\t\t\t-> show actual setting of rpc (disabled/enabled) (on server side)\n"\
        "--disable_rpc\t\t\t-> disable rpc calls (on server side)\n"\
        "--enable_rpc\t\t\t-> enable rpc calls (on server side)\n"\
        "--shutdown\t\t\t-> shutdown server\n"\
        "--change_interval <int>\t\t-> change parsing interval of server\n"\
        "--keyword_status\t\t-> show actual setting of \"keywords\" (on server side)\n"\
        "--add_keyword <string>\t\t-> add keyword to keyword list (on server side)\n"\
        "--delete_keyword <string>\t-> delete keyword to keyword list (on server side)\n"\
        "--ignore_error_status\t\t-> show actual setting of \"ignoer_error\" (on server side)\n"\
        "--add_ignore_error <int>\t-> add error, which the parser should ignore (on server side)\n"\
        "--delete_ignore_error <int>\t-> delete error, which the parser is ignoring (on server side)\n"
    if color == 1:
        msg += color_obj.green("\n-----  additional parameters  -----\n")
    else:
        msg += "\n-----  additional parameters  -----\n"
    msg += "\n--ip <ip>\t\t\t-> host IP\n"\
        "--port <int>\t\t\t-> port, where the server is listening\n"
    if color == 1:
        msg += color_obj.green("\n-----  examples  -----\n\n")
        msg += color_obj.blue("python gui.py -c config_gui.ini --disable_rpc --ip 127.0.0.1 --port 6000\n")
        msg += color_obj.yellow("\t-> disable rpc calls on localhost\n\n")
        msg += color_obj.blue("python gui.py -c config_gui.ini --ip 127.0.0.1 --port 6000 --add_keyword status:\!error\n")
        msg += color_obj.yellow("\t-> add new keyword set \"status AND NOT error\" into keyword file on localhost\n\n")
    else:
        msg += "\n-----  examples  -----\n\n"\
                "python gui.py -c config_gui.ini --disable_rpc --ip 127.0.0.1 --port 6000\n\t-> disable rpc calls on localhost\n\n"\
                "python gui.py -c config_gui.ini --ip 127.0.0.1 --port 6000 --add_keyword status:\!error\n\t-> add new keyword set \"status AND NOT error\" into keyword file on localhost\n\n"
    msg += "\n"

    return msg



#############################################################

def start():

    '''
    The functions starts the application.
    '''
    col = 1
    rpc_status = None
    disable_rpc = None
    enable_rpc = None
    shutdown = None
    change_interval = None
    interval_status = None
    keyword_status = None
    add_keyword = None
    delete_keyword = None
    add_error = None
    delete_error = None
    error_status = None
    ip = None
    port = None
    # parameter evaluation
    try:
        opts, args = getopt.getopt(sys.argv[1:], 'c:hg', ['config=', 'nocolor', 'help
            ', 'rpc_status', 'disable_rpc', 'enable_rpc', 'shutdown', '
            interval_status', 'change_interval=', 'keyword_status', 'add_keyword=', '
            delete_keyword=', 'ignore_error_status', 'add_ignore_error=', '
            delete_ignore_error=', 'ip=', 'port='])
        for opt, value in opts:
            if opt in ('', '--nocolor'):
                col = 0
            if opt in ('-h','--help'):
                msg = help_context(col)
                usage_exit(sys.argv[0], msg, col)
            if opt in ('-c', '--config'):
                value = value.replace("=", "")
                configfile = os.getcwd()+"/"+value

        for opt, value in opts:
            if opt in ('', '--rpc_status'):
                rpc_status = 1
            if opt in ('', '--disable_rpc'):
                disable_rpc = 1
            if opt in ('', '--enable_rpc'):
                enable_rpc = 1
            if opt in ('', '--shutdown'):
                shutdown = 1
            if opt in ('', '--interval_status'):
                interval_status = 1
            if opt in ('', '--change_interval'):
                change_interval = int(value)
            if opt in ('', '--keyword_status'):
                keyword_status = 1
            if opt in ('', '--add_keyword'):
                add_keyword = value
            if opt in ('', '--delete_keyword'):
                delete_keyword = value
            if opt in ('', '--add_ignore_error'):
                add_error = int(value)
            if opt in ('', '--delete_ignore_error'):
                delete_error = int(value)
            if opt in ('', '--ignore_error_status'):
                error_status = 1
            if opt in ('','--ip'):
                ip = value
                status = re.search('^[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}', ip
                    )
                if (None == status):
                    usage_exit(sys.argv[0], 'given IP is not valid', col)
                else:
                    ip = status.string[status.start():status.end()]
                    if (0 != check_ip(ip)):
                        usage_exit(sys.argv[0], 'given IP is not valid', col)
            if opt in ('', '--port'):
                port = int(value)
                if (port < 1024 or port > 50001):
                    usage_exit(sys.argv[0], "Server port is out of range! \nMake sure the server port lies between 1025 (inclusive) and 50000 (inclusive)!\n\n", col)
    except getopt.error, e:
        e = "%s" % e
        usage_exit(sys.argv[0], e, col)
    except ValueError, e:
        e = "%s" % e
        usage_exit(sys.argv[0], e, col)

    # load config file or default values
    if (configfile != ""):
        # check if file exists
        if(1 == os.path.exists(configfile)):
            config = LoadConfig(configfile)
        else:
            # if file NOT exists terminate program
            print "\n\nSorry, a given config file does NOT exist !\nPlease try again !\n\n"
            os._exit(-1)
    else:
        msg = "\nNo config file spezified !\n"
        usage_exit(sys.argv[0], msg, col)


    gui = Admin(config)

    if col == 1:
        col_obj = Colour()

    #---------------------- SERVER COMMANDS ----------------------#

    if (1 == rpc_status):
        # get rpc status on server side
        if (None != ip and None != port):
            text = "Check RPC status on server \"%s:%d\"." % (ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_status()
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (1 == disable_rpc):
        # disable_rpc on server side
        if (None != ip and None != port):
            text = "Disable RPC on server \"%s:%d\"." % (ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.disable_rpc_calls()
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (1 == enable_rpc):
        # enable_rpc on server side
        if (None != ip and None != port):
            text = "Enable RPC on server \"%s:%d\"." % (ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.enable_rpc_calls()
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (1 == shutdown):
        # shutdown the server
        if (None != ip and None != port):
            text = "Shutdown server \"%s:%d\"." % (ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.stop_server()
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (None != change_interval):
        # change parsing interval time
        if (None != ip and None != port):
            text = "Change parsing interval on server \"%s:%d\" to %d minutes." % (ip
                , port, change_interval)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_configuration("misc", "minute",
                    change_interval, 2)
                if answer == 0:
                    answer = "interval successfully to %d minutes changed" %
                        change_interval
                else:
                    answer = "could not change interval \n-> \"%s\"" % answer
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text
    elif (1 == error_status):
        # get ignore error status from server
        if (None != ip and None != port):
            text = "Get status for \"ignore_error\" from server \"%s:%d\"." % (ip,
                port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_configuration("misc", "ignore_error",
                     0, 4)
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (None != add_error):
        # add ignore error
        if (None != ip and None != port):
            text = "Add \"ignore_error\" %s on server \"%s:%d\"." % ( add_error, ip,
                port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_configuration("misc", "ignore_error",
                     add_error, 1)
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (None != delete_error):
        # delete ignore error
        if (None != ip and None != port):
            text = "Delete \"ignore_error\" %s on server \"%s:%d\"." % ( delete_error
                , ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_configuration("misc", "ignore_error",
                     delete_error, 0)
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (1 == keyword_status):
        # get keywords which are used currently
        if (None != ip and None != port):
            text = "Get keywords from server \"%s:%s\"." % ( ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_keyword_file("status", 2)
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (None != add_keyword):
        # add new keyword
        if (None != ip and None != port):
            text = "Add keyword \"%s\" on server \"%s:%s\"." % (add_keyword, ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_keyword_file(add_keyword, 1)
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (None != delete_keyword):
        # delete keyword
        if (None != ip and None != port):
            text = "Delete keyword \"%s\" in keyword list on server \"%s:%s\"." % (
                delete_keyword, ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_update_keyword_file(delete_keyword, 0)
                if col == 1:
                    answer = col_obj.green(answer)
                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text

    elif (1 == interval_status):
        # get current parsing interval time
        if (None != ip and None != port):
            text = "Get parsing interval time (in minutes) from server \"%s:%s\"." %
                (ip, port)
            if col == 1:
                text = col_obj.yellow(text)
            print
            print text
            serv_object = gui.connect_to_server(ip, port)
            try:
                answer = serv_object.rpc_interval_status()
                if answer != -2:
                    answer = "every %d minutes" % answer
                else:
                    answer = "RPC disabled"
                if col == 1:
                    answer = col_obj.green(answer)

                print "\nserver -> %s" % answer
            except AssertionError:
                text = "Server is down !"
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
            except:
                text = "Could not connect to server \"%s:%d\"." % (ip, port)
                if col == 1:
                    print col_obj.red(text)
                else:
                    print text
        else:
            text = "\nNo IP or port given !\n"
            if col == 1:
                print col_obj.red(text)
            else:
                print text
    else:
        text = "No command given !\nUse option -h or --help to display the help."
        usage_exit(sys.argv[0], text, col)


if __name__ == '__main__':

    start()
```

D.5 GZ Parser

LISTING D.13: Module gz_parser.py

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

'''
This is the gz_parser.py module, which uses an external config file (e.g. config_gz_parser.ini) to parse through a directory with *.gz files. The server_classes.py is also needed.

Reading University
MSc in Network Centered Computing
a.weise - a.weise@reading.ac.uk - December 2005
'''

import os, sys, string, re, stat
from server_classes import LogFileParser
import ConfigParser, getopt

gz_list = [] #save *.gz files


def LoadConfig(file, config={}):
    """
    This functions returns a dictionary with key's of the form
    <section>.<option> and the values .

    source: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/65334
    """
    config = config.copy()
    cp = ConfigParser.ConfigParser()
    cp.read(file)
    for sec in cp.sections():
        name = string.lower(sec)
        for opt in cp.options(sec):
            config[name + "." + string.lower(opt)] = string.strip(cp.get(sec, opt))
    return config

def parse_directory(arg, dirname, fnames):
    '''
    This function "walks" through a given directory and considers all srbLOG*.gz files. The name and last modified time are saved in a list (2 dimensional array). The function should be used with os.path.walk(path, function_name, arg)!
    '''
    d = os.getcwd()
    # change into log file directory
    try:
        os.chdir(dirname)
    except:
        print "could not find directory \"%s\"" % dirname
        return -1
    # for each file
    for f in fnames:
        # check if file and if file is a log file e.g. srbLog.20051003.gz
        if (not os.path.isfile(f)) or (None == re.search('^srbLog[_0-9.-]*.gz', f)):
            continue
        # get last modified time
        date = os.stat(f)[stat.ST_MTIME]
        # create tupel
        tupel = (date, f)
        # save last modified time and filename into am arrray (list)
        gz_list.append(tupel)
    # change back into the working directory
    os.chdir(d)

def get_keywords(filus):
    '''
    This function extracts keyword from a give file!
    '''
    keys = []

    try:
        file_fd = file(filus, 'r')
    except IOError, e:
        print "Problem with keyword file -> ", e
        return -1

    content = file_fd.readlines()# save file content as list (1 line == 1 entry)

    file_fd.close()

    content = remove_item(content, "#") # remove comments
    content = remove_item(content, "\n")# remove linebreaks

    for i in range(len(content)):
        content[i] = content[i].strip()
        content[i] = content[i].rstrip(",")
        content[i] = content[i].split(",")
        for a in range(len(content[i])):
            keys.append(content[i][a])

    for i in range(len(keys)):
        keys[i] = keys[i].strip() # remove whitespace
        keys[i] = keys[i].split(":")

    return keys

def remove_item(listus, item):
    '''
    This function removes "items" form a list object rekursiv.
    '''

    while(1):
        for i in range(len(listus)):
            if -1 != listus[i].find(item, 0, 1):
                del listus[i]
                remove_item(listus, item)
                break
        else:
            break

    return listus

def gunzip(filus, name_temp_file="temp_srbLog"):
    '''
    This function unzips a *.gz file using the system tool gunzip. Make sure when
        calling the function the file exists in this directory. The function creates
        a temporary file and leave the orignal *.gz file untouched!
    '''
    if (not os.path.isfile(filus)):
        return -1
    else:
        command = "gunzip -c %s > %s" % (filus, name_temp_file)
        os.system(command)
        return 0

def delete_file(filus):
    '''
    This functions deletes a given file.
    '''
    try:
        os.remove(filus)
        return 0
    except:
        print "could not delete -> ", filus
        return -1

def usage_exit(progname, msg=None):
    '''
    This function displays the usage of the program and terminated the script.
    '''
    if msg:
        print msg
        print
    print "usage: %s -h|--help -c|--config -v|--verbose " % progname
    os._exit(-1)

##########################################################################

def start():
    '''
    This function starts the application.
    '''
    global gz_list
    gz_list = [] #save *.gz files
    configfile = ""
    verbose = 0

    # evaluate parameters
    try:
        opts, args = getopt.getopt(sys.argv[1:], 'c:vh', ['config=', 'verbose', 'help
            '])
        for opt, value in opts:
            if opt in ('-h','--help'):
                msg = "Help:\n-c or --config\t->\tdefines config file, if no config
                    file given, default values are used\n-v or --verbose\t->\
                    tactivates printing of messages [debug option]\n-h or --help\t->\
                    tprints this help"
                usage_exit(sys.argv[0], msg)
            elif opt in ('-c','--config'):
                value = value.replace("=", "")
                configfile = os.getcwd()+"/"+value
            elif opt in ('-v','--verbose'):
                verbose = 1
            else:
                usage_exit(sys.argv[0], "Wrong use of parameter")
    except getopt.error, e:
        usage_exit(sys.argv[0], e)

    # load config file or default values
    if (configfile != ""):
        # check if file exists
        if(1 == os.path.exists(configfile)):
            config = LoadConfig(configfile)
        else:
            # if file NOT exists terminate program
            print "Sorry, a given file does NOT exist !\nPlease try again!\n\n"
            os._exit(-1)
    else:
        msg = "\nNo config file spezified !\n"
        usage_exit(sys.argv[0], msg)

    print "\n\n------ GZ SRB LOG FILE PARSER ------"

    workingpath = os.getcwd()

    path_srb_gz = config.get("path.path_srb_gz")
    path_srb_gz = path_srb_gz.rstrip("/")
    path_xml_file = config.get("path.path_xml_file")
    path_xml_file = path_xml_file.rstrip("/")
    xml_file_name = "gz_client_log.xml"

    # check if the configuration is correct
    if(0 == os.path.exists(path_srb_gz)):
        print "Could not locate log file archive path under %s !\nMaybe change
            configuration file and try again!\n\n" % path_srb_gz
        os._exit(-1)

    if(0 == os.path.exists(path_xml_file)):
        print "Could not locate xml path under %s !\nMaybe change configuration file and try again!\n\n" % path_xml_file
        os._exit(-1)

    keyword = config.get("file.keyword")
    keyword_path = config.get("path.path_keyword")
    if keyword != None:
        keyword = keyword.strip()
    if keyword_path != None:
        keyword_path = keyword_path.rstrip("/")
    if(keyword_path == '' or keyword_path == None):
        keyword_path = workingpath
    else:
        if (-1 != keyword_path.find("/", 0, 1)):
            # first character ”/”
            pass
        else:
            keyword_path = workingpath+"/"+keyword_path

    keyword_list = get_keywords(keyword_path+"/"+keyword)

    ignore_error = config.get("misc.ignore_error")
    if ("" != ignore_error):
        ignore_error = ignore_error.split(",")
        for i in range(len(ignore_error)):
            ignore_error[i] = int(ignore_error[i].strip())

    parserus = LogFileParser(path_srb_gz, keyword_list, ignore_error, os.getcwd(), "temp_client_log.xml", verbose)

    os.path.walk(path_srb_gz, parse_directory, gz_list)
    d = os.getcwd()
    os.chdir(path_srb_gz)
    if (0 < len(gz_list)):
        try:
            for x in range(len(gz_list)):
                print "\n"
                print x+1,
                print ". parsing -> \"%s\"\n" % gz_list[x][1]
                gunzip(gz_list[x][1])
                status = os.stat(gz_list[x][1])
                parserus.analyse_log_file("temp_srbLog", file_time=status[8])
                delete_file("temp_srbLog")
        except:
            os.remove("temp_srbLog")
            os.chdir(d)
            os.remove("temp_client_log.xml")
            print "Problem parsing log files -> terminating !"
            os._exit(0)
    else:
        print "Could not find any srbLog*.gz files!"
        os._exit(0)

    os.chdir(d)


    test_file = "%s/%s" % (path_xml_file, xml_file_name)

    # check if a gz_client_log.xml already there, if yes change name
    c = 1
    while(1):
        if(0 == os.path.exists(test_file)):
            break
        test_file = "%s/%d_%s" % (path_xml_file, c, xml_file_name)
        c += 1

    print "\ncopy xml file ..."
    command = "cp temp_client_log.xml %s" % test_file
    os.system(command)

    # delete temporary xml file
    delete_file("temp_client_log.xml")

    print "\n\ndone ... \n\n"

if __name__ == '__main__':
    start()
```

Appendix E

CD ROM

```
Content
    └ Monitoring Tools
            └ readme.txt
            └  Server
            │       └start_server.py
            │       └server_classes.py
            │       └utils_server.py
            │       └ stop_server.sh
            └  Client
            │       └start_client.py
            │       └client_classes.py
            │       └utils_client.py
            │       └ stop_client.sh
            └  Virtualiser
                    └  gui.py
                    └ gui_classes.py
                    └ gui_utils.py
                    └ Graphs.py
                    └  utils.py
                    └ tooltips.py
```

Appendix F

Publications

The following paper was presented at the 2006 SDSC SRB Workshop. *The 2006 SDSC SRB Workshop was a forum for SRB user community researchers and practitioners to share their knowledge, experiences, and solutions in utilizing this technology, to gain additional insight into SRB configurations, techniques, and options, and to provide feedback to, and hear of future development plans from, the SRB team.* [51] *It was held February 2nd and 3rd at SDSC in San Diego.* [51]

Some Tools for Supporting SRB Production Services

R. Downing
CCLRC-Daresbury Laboratory

A. Weise, C. Koebernick
University of Reading

A. Hasan
CCLRC-Rutherford Appleton Laboratory

Abstract

Providing production-level services requires monitoring applications, performance and intercepting errors as soon as they occur. In this paper we describe some of the tools that have been developed to assist production SRB services. We describe the approaches used and how they can be more generally applied.

1. Introduction

The Data Management Group (DMG)[1] is part of the Council for the Central Laboratory of the Research Councils (CCLRC) e-science centre [2] and provides data storage solutions for a large number of e-science projects. The DMG uses the Storage Resource Broker (SRB) [3] as the core component for many projects, tailoring the system to meet the needs of the project. Once a system is deployed the DMG also provides a level of support for the service ranging from troubleshooting to responding to further feature requests and upgrades.

Through the course of developing various SRB systems we have managed to identify a number of tasks that appear common and which greatly help in supporting a production system. In this paper we describe a few of the tools developed to aid this task.

2. Monitoring Production Servers

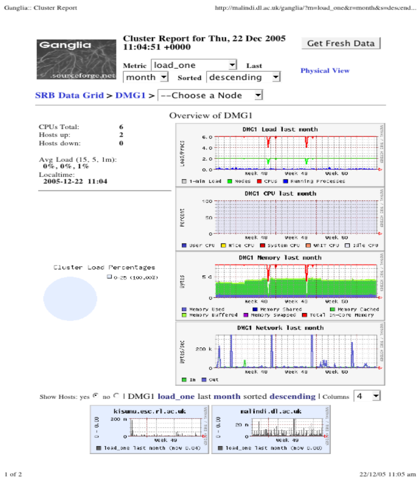
Careful monitoring of production servers provides a number of benefits: aids debugging, provides information on the distribution of load in the system and provides information for planning purposes. Troubleshooting and load balancing require both instantaneous information and also historic information whereas planning requires only historic information.

2.1. Ganglia and Nagios Monitoring

Since the SRB system is distributed any monitoring application must be capable of working with distributed systems. With this requirement in mind we have selected Nagios [4] to report instantaneous information on server properties, such as cpu, machine load, etc. The Nagios system emails a list of subscribers when any of the monitored properties of a server go beyond an acceptable threshold limit as well as reporting when a server is down.

For the collection of historic information we chose Ganglia [5]. The Ganglia monitoring system collects a set of system properties at regular intervals and stores them in a round-robin database. It is also possible to monitor additional properties by providing a script to extract these properties to Ganglia. The system also provides tools for presenting the information as a series of web-pages (see figure 1). As we run more than one SRB server on a given host we needed to make a minor kludge to allow the same host to appear in more than one group.

[*] This work has been funded by a range of UK agencies incl. the e-Science Programmes of the Natural Environmental Research Council, the Engineering and Physical Science Research Council, the Council of the Central Laboratory of the Research Councils, the Biotechnology and Biological Sciences Research Council and the Joint Information Systems Committee.



Ganglia:: Cluster Report http://malindi.dl.ac.uk/ganglia/?m=load_one&r=month&s=descend...

Cluster Report for Thu, 22 Dec 2005 11:04:51 +0000

Metric load_one Last month Sorted descending

SRB Data Grid > DMG1 > --Choose a Node

Overview of DMG1

CPUs Total:	6
Hosts up:	2
Hosts down:	0

Avg Load (15, 5, 1m):
0%, 0%, 1%
Localtime:
2005-12-22 11:04

Show Hosts: yes ● no ○ | DMG1 **load_one** last **month** sorted **descending** | Columns 4

1 of 2 22/12/05 11:05 am

Figure 1: Ganglia web page displaying usage for a test SRB server.

3. Monitoring SRB Server Log Files

Each SRB server writes activity information to a log file. These log files contain information about which process, and from which machine, connected to the SRB server as well as error messages detected by the server when handling a request. These error messages along with the time that they occurred are an essential tool in troubleshooting. It is important to notify administrators as soon as an error occurs, it is also important to log the error messages in order to identify chronic problems and possibly identify patterns.

Any application to monitor the log files would need to be able to parse the log files for error messages, email to a subscriber list serious errors and collect in a central location the error messages for later searches. With these requirements we decided to build a system in Python to parse, log and notify when error messages occurred [6].

It is possible that Ganglia could be used to parse the log files and store the resulting error messages in a central round-robin database, but we found that the database was not flexible enough for our queries and we also required email notification when problems occurred.

The system essentially consists of three components: a Parser a Collector and a Displayer, figure 1 shows a simple diagram of how the application works.

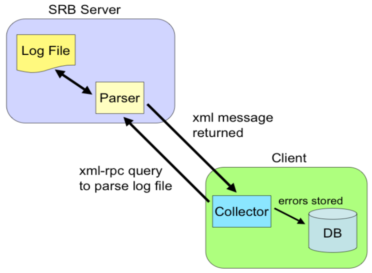


Figure 2: A simple schematic showing the log file parser system.

The Parser is actually an XML-RPC server that is started on the SRB server host and consists of a method to parse the SRB log file. The Collector is a daemon that sends an XML-RPC message to the Parser to parse the log file. The parser then returns an XML message containing the error message, line number, date, server and error message code to the Collector. The Collector then extracts the information from the XML message and stores the contents in an SQLite database and sends an email containing the error message information to a list of subscribers. The list of SRB servers that the Collector should contact and the frequency with which to contact them is read from a configuration file.

The Displayer is used to graphically display the error messages as a function of server that can help in identifying potentially chronic problems with a server. The Displayer can also plot error messages of a particular type as a function of time that may reveal interesting patterns that could help troubleshooting. Figure 2 shows a screenshot of a histogram of error messages for a given server.

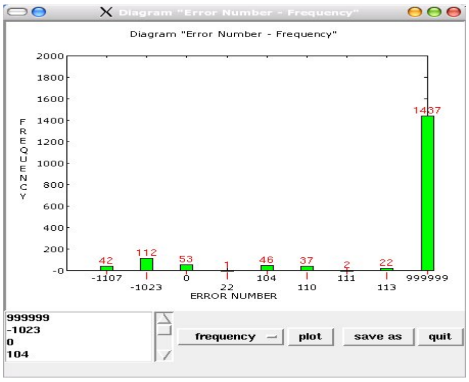


Diagram "Error Number - Frequency"

```
999999
-1023
0
104
```

Figure 3: Screenshot of the error message numbers extracted from an SRB log file.

The numbers above the bars correspond to the actual occurrences of errors with that error number and error number 999999 corresponds to messages that do not have an SRB defined error number.

The Parser assumes all messages are error messages unless the user specifies in a configuration file a pattern contained in messages that should be ignored. The approach of assuming every message is an error ensures that we do not accidentally miss an unusual error message.

4. Tools for Measuring Performance

Measuring the performance of a system is important as it helps to determine the capabilities of the system, it helps to determine bottlenecks in the system and it provides a means of tuning a system. We have developed a framework that can be used to run performance tests [7] and a number of scripts that execute performance tests using Scommands on an SRB system.

The framework consists of the Ganglia monitoring system to monitor the SRB server and client application, an SQLite database to hold the measurements and Collector collect the results from Ganglia and store them in the SQLite database. The framework can also display, in real-time, graphs of the server properties as a function of time. A Displayer is also provided to graphically display previous data with the option to overlay previous performance tests. Figure 3 shows a simple schematic of the framework.

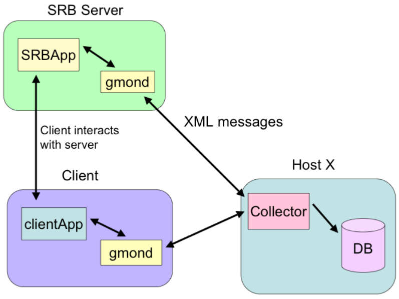


Figure 4: Schematic of the framework for performance measurements.

The Ganglia gmond daemons on the client and server machine are started by the Collector daemon before the performance tests start. The Collector collects the monitoring information in the form of XML messages at periodic intervals, extracts the information from the XML message and stores it in the SQLite database.

At this point the client application can be started and the performance measurements are recorded. The Collector reads from a configuration file the host names and applications that should be monitored as well as the interval at which the data should be collected. Figure 4 shows the cpu-load graph produced by the Displayer.In principle, the framework is not tied to the SRB and can be used for any application.

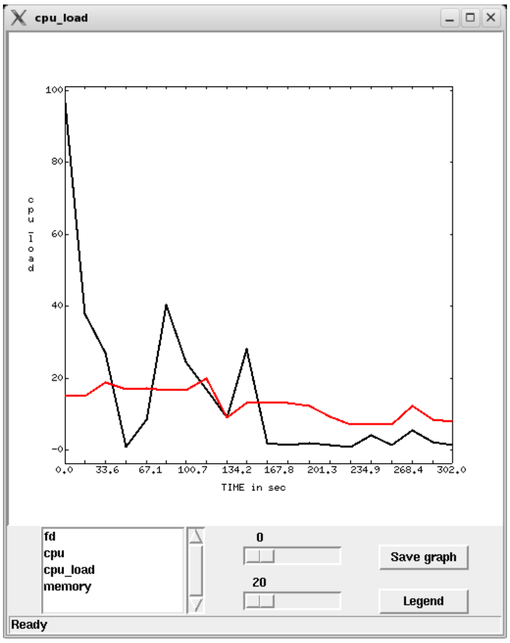


Figure 3: Graph of cpu-load produced by the Displayer application.

In order to measure the performance of an SRB system we have developed a set of tools based on the Scommands. The tools are capable of storing information in the SRB as collections, containers or simply files. The tools are configurable and can store large numbers of objects in flat or nested directory structures and are also capable of producing nested collections. The tools can also store variable amounts of metadata within the SRB.

5. Conclusion

Monitoring a production system is an essential aid in planning future extensions to the system, it can also be an essential aid in troubleshooting. Tools to carry out performance tests and collection, store and present the data are also important as they provide a means of providing a references against which the production system performance can be measured. Such tools can also help in troubleshooting problems either by comparing the performance against a benchmark, or simply by exercising a particular aspect of the system.

In this paper we have described a few of the tools that we have developed to help our production systems. All the tools we have developed are extensible as they have to accommodate new features or aspects of the production system.

References

[1] http://www.e-science.clrc.ac.uk/web/groups/Data-Management/Data-Management
[2] http://www.rcuk.ac.uk/escience
[3] http://www.sdsc.edu/srb
[4] http://www.nagios.org
[5] http://ganglia.sourceforge.net
[6] A. Weise, *M.Sc Thesis (in preparation).*
[7] C. Koebernick. *M.Sc Thesis (in preparation).*

Appendix G

Declaration of Authorship

I declare that this dissertation is my own, unaided work, except where otherwise acknowledged or referenced. It is being submitted for the degree of Master of Science at the University of Reading.

It has not been submitted before for any other degree or examination in any other university.

Reading, 4th March 2006

Andrea Weise